MASTER IN COMPUTER SCIENCE

Delivering protected content
an approach for next generation mobile technologies

Collet, Jean Bernard

Award date:
2010

Awarding institution:
University of Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Faculté d'Informatique
Anne académique 2009-2010

Delivering protected content : an approach for next generation mobile technologies

Jean Bernard COLLET

Mémoire présenté en vue de l'obtention du grade de master en informatique.

Abstract

The mobile industry is growing every day and the mobile phone became in the past few years the most personal device ever. Often, it is the first interface someone looks at in the morning (with the alarm clock) and the last one in the evening (either for the alarm clock or just check the last messages). The combination of this personal aspect and its growing power opens new commercial perspectives for the future. It is easy now to imagine watching movies in the train via streaming or side loading using those next generation mobile devices. This experience can even be dramatically improved by using adaptive streaming, a technology that allows for dynamic selection and adaptation of video quality during playback, based on the available bandwidth, thus providing an overall better end-user experience. However, delivering digital content to mobile devices raises the copyright and intellectual property issues. Therefore, protected delivery mechanisms are required, that often rely on a specific DRM (Digital Rights Management) scheme.

In this thesis, we will address design and implementation of a DRM agent on an open platform in order to support protected media when using adaptive streaming.

Keywords

computer science, Android, DRM, adaptive streaming, secure storage, post-instalable

Résumé

L'industrie de la téléphonie mobile est en constante croissance et le téléphone mobile est devenu depuis quelques années l'appareil le plus personnel jamais créé. Souvent, c'est la première interface informatique qu'une personne consulte le matin (avec le réveil de l'appareil) et la dernière (pour ajuster le réveil de l'appareil, ou simplement consulter ses messages). La combinaison de cet aspect personnel et de la puissance en augmentation de ces appareils ouvre de nouvelles perspectives commerciales pour l'avenir. Il est très simple d'imaginer regarder un film dans le train en streaming ou après avoir pré-téléchargé le contenu. L'expérience utilisateur peut être fortement améliorée grâce à l'utilisation de l'adaptive streaming, une technologie qui permet la selection dynamique et l'adaptation de la qualité video (et audio) pendant la lecture d'un contenu (estimé en fonction de la bande passante). Cela offre alors une expérience utilisateur bien meilleure. Cependant, fournir du contenu digital à des smartphones soulève des problèmes de droits d'auteur. Des mécanismes de protection sont nécessaires. Ceux-ci sont souvent basés sur les DRM (Digital Rights Management).

Dans ce mémoire, nous allons nous concentrer sur le design et l'implémentation d'un agent DRM sur une plateforme ouverte afin d'utiliser de l'adaptive streaming.

Mots-clés

informatique, Android, DRM, adaptive streaming, stockage sécurisé, post-installable

Acknowledgements

This thesis is the final result of five-year of university in Computer Science. It is a important step in my life between university and a future job. It is also the result of many hours of reading, researches, tests, developments and writing. This work was also a great opportunity to meet very interesting people and places around Europe.

First, I want to return thanks to Prof. Jean-Noël Colin, my promotor, for its support and its feed-backs. He guided me for my internship in Germany. We had very interesting discussions about how to write this thesis, and about its related technologies. It was a real pleasure to work together. I hope we will have further opportunities to work together again and wish him the best for the future.

Then, I also want to return thanks to everyone who helped me during this work like Doc. Susanne Guth, Boras Fehr and Michael Stattmann. They provided me a very great help and support during my internship and thanks to them, I had the chance to be introduced into German culture.

Finally, I want to return thank to my family and particularly Sophie Dupuis, for her support, her advices, her trust and simply for being present.

Table of Contents

List of Figures

List of Listings

Chapter 1

Introduction

1.1 Context and objectives of the thesis

Vodafone has become in the past few years, more than a telecommunication network operator. It is now a major service provider (media content provider, social network) and works with various networks (mobile networks and broadband) and devices types (desktop, laptop, netbooks, smartphones). Due to this variety of devices, they have to deal with a wide market, and therefore, they have to deploy products adapted for many platforms and types of users.

Nowadays, most of the content providers require the copyrighted content to be protected via secured system. The most common one, even if it is unpopular, is Digital Rights Management. DRM provides to the content provider mechanisms to control the use of digital media content like music, movies, games and many other types of content. As business models are changing every day, this technology is still evolving and need to be upgraded frequently.

The purpose of this thesis is to describe in detail a method to implement and deploy a new way to deliver and consume protected content for modern smartphones, which are the next generation mobile devices. It will detail the complete implementation of the media player, the integration of DRM's, the content preparation and the server deployment.

1.2 Motivations

Actually, on most mobile devices, the Open Mobile Alliance (OMA) DRM is the de facto standard implemented by default. This implementation is installed during the manufacturing process where *Vodafone* has no or limited access to. Some DRM Client located on those devices may only have a partial support of the standard i.e. some features are missing.

The reason why most DRM Client are installed on devices while manufacturing process is because they need particular features from the operating system to enable a good

security level. For example, the necessary key certificates are pre-installed that way instead of Over The Air (OTA) installation. Other particular features may be a specific filesystem for sensitive data storage, mechanisms to authenticate an application, secure channels for data exchange. Most of the time, those features need some system modifications.

Everyday, the market evolves and new business models appear. And they may require particular features that may not be supported by a DRM Client. This is one of many reasons for using post-installable DRM Client on device. Sometimes, existing standards need to be adapted depending on the new feature to be implemented. Those adaptations are needed because when a company finds a new market to explore, they have to react as fast as possible. They could not afford to wait for standards to be compliant with their needs. Time to market is thus an determinative factor.

The present work will use a technique called "Adaptive Streaming" for media consumption. Its particularity is it improves the user experience while the video playback by enabling video quality switch. It results on a video quality which perfectly match the bandwidth capacity and no lags while playback. Used on a device with a fully OMA DRM compliant DRM Client, it would not work due to the fact that OMA DRM currently does not support that technology. In that particular case, an adaptation should not be so complex to do, but will still need to extends the existing standard.

The previous example demonstrate the need of a post-installable DRM client. The following work will try, among other things, to set up concepts and design lines for building a post-installable content protection solution integrate them on the proof of concept of this work. That solution must meet some requirements like *security*, *easy to evolve*,*multiple platform compliant*, *multiple standard compliant* and *easy to maintain*.

- *Security :* Security is a key aspect of content protection, simply because every protection mecanism depends on it. It would require to study, among others, authentication mechanisms (to check if the program is really who he claims to be), cryptography and key distribution (for encrypting files that should not be read by other applications) and filesystems (to store protected data in a secure way).
- *Easy to evolve :* As mentioned earlier, an easy to evolve solution is also a key aspect. As the market and the needs change every day, mechanisms and support for adapting existing standards are necessary to provide a solution in the long term.
- *Multiple platform compliant :* Nowadays, the mobile phone market is changing. Many operating systems are coexisting and are often updated. Thus, the solution described in this document should be as platform independent as possible. However, some system specifications may be required.
- *Multiple standard compliant :* Due to the fact that a lot of standards are actually existing and used, it is important too to have a solution which can switch easily from one standard to another, or to one version to the next.
- *Easy to maintain :* All of the previous described aspects tend to evolve. Maintenance is also really important for correcting bugs first, then to tend to apply major updates to every device, no matter its platform and standard.

1.3 Overview of the thesis

The first part on this work will focus on the technological context. It will explain the basic concepts of the technologies and techniques used, such as Google's Android, Digital Rights Management and Adaptive Streaming.

Then, the solution design will be explained, starting with the use cases of the proof of concept produced with this work, then the operating constraints will be introduced. Every single component of this project will be detailed.

After explaining the solution design, the implementation will be introduced with a technical point of view. Experiments will be done and commented.

Finally, we will conclude this work with further work, improvements and general comments on the feasibility of this project.

1.4 Notations

1.4.1 Definitions

Content	One or more objects
Content Issuer	The entity making content available to the DRM Agent
DRM Agent	The component in the device that manages Permissions for Media Objects on the device.
DRM Content	Media Objects that are consumed according to a set of Permissions in a Right Object
Media Object	A digital work e.g. a ringtone, a screen saver, a game or a composite object
Permission	Actual usages or activities allowed (by the Right Issuer) over DRM Content.
Right Issuer	An entity that issues Rights Objects to OMA DRM conformant devices.
Rights Objects	A collection of Permissions and other attributes which are linked to DRM Content.
Super-distribution	A mechanism that allows a User to distribute DRM Content to other devices through potentially insecure channels and enables the User of that device to obtain a Right Object for the super-distributed DRM Content.

1.4.2 Abbreviations

CEK	Content Encryption Key
CMLA	Content Management License Administrator
CPU	Central Processing Unit
DCF	DRM Content Format
DRM	Digital Right Management
GPS	Global Positioning System
GSM	Group System for Mobile
GOP	Group Of Pictures
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IrDA	Infrared Data Association
JNI	Java Native Interface
MMS	Multimedia Messaging Service
NDK	Native Development Kit
OCSP	Online Certificate Status Protocol
OMA	Open Mobile Alliance
OS	Operating System
OTA	Over The Air
PKI	Public Key Infrastructure
RDT	Real Data Transport
REK	Right Encryption Key
RFC	Request For Comments
RI	Right Issuer
ROAP	Right Object Acquisition Protocol
RO	Right Object
RTP	Real Time Transport Protocol
RTSP	Real Time Streaming Protocol
SDK	Software Development Kit
SMS	Short Message Service
UI	User Interface
URI	Uniform Resource Indicator
VM	Virtual Machine
WAP	Wireless Application Protocol

Chapter 2

Technological context

This section's purpose is to describe the key technologies to support the solution designed during this work. First of all, it starts with a short description of Android, a mobile operating system made by Google. The second part will be about Digital Right Management (DRM) with an overview of existing standards and a brief comparison. The next part will introduce the existing video technologies, especially streaming and streaming at variable bitrate. Finally, some security principles will be explained in order to understand further developments.

2.1 Mobile Operating System : Google's Android

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. It includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple Virtual Machines efficiently. The Dalvik Virtual Machine executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management. Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

2.1.1 Application development

On Android, applications are written in a Java like language. To compile such software, a developer has to use the Software Development Kit (SDK) for the target Android platform (1.5, 1.6, 2.0, 2.1, 2.2, ...) provided by Google. It is also possible to enrich softwares written with the SDK with "native" libraries, written in C and interfaced with the Java application through Java Native Interface (JNI) calls. In order to use native code (C/C++ code) and JNI, the Native Development Kit (NDK) is required. It's not possible to write an entire application with the NDK but only libraries which will have

to be included into a SDK project.

Every application is compiled and assembled by the SDK into an Android Package, an archive file marked with the ".apk" suffix. This archive contains every resources (compiled source code, images, layouts, compiled native libraries...) an application needs (see Fig.2.1).

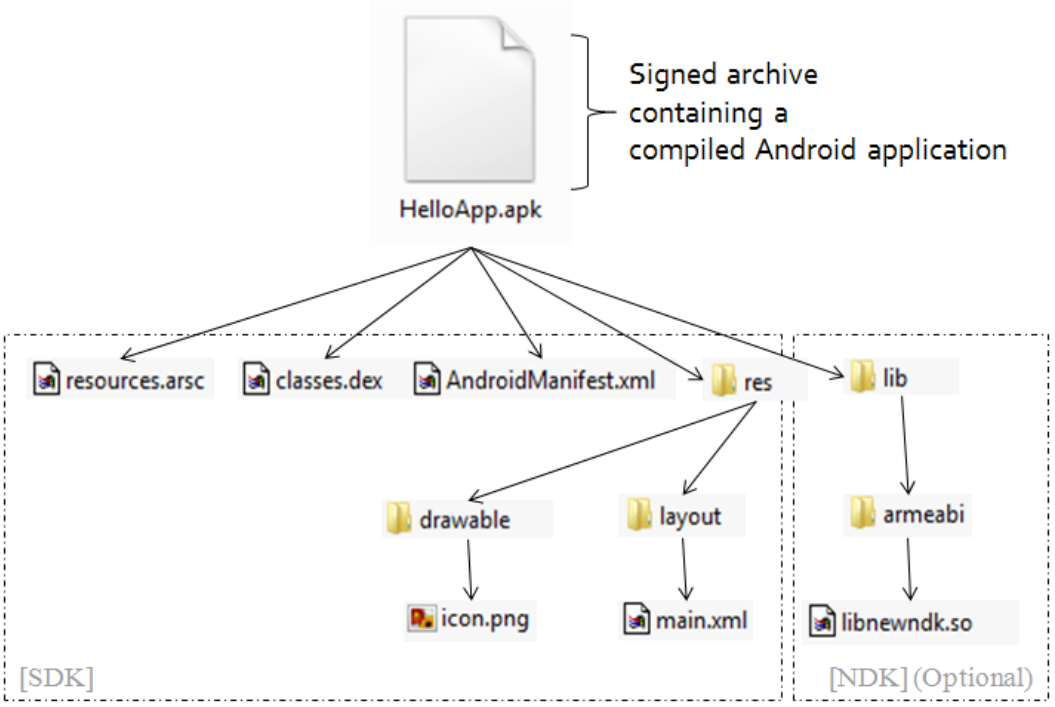

FIGURE 2.1 – Anatomy of an Android application

Applications can combine components of four different types : Activity, Service, Broadcast Receiver and Content Provider.

Activity

An Activity presents a visual user interface for one focused endeavor the user can undertake. For example, a text messaging application might have one activity that shows a list of contacts to send messages to, a second activity to write the message to the chosen contact, and other activities to review old messages or change settings. Of course, some application will need only one single activity.

Typically, one of the activities is marked on the application's manifest as the first one that should be presented to the user when the application is launched. Moving from one activity to another is accomplished by explicitly asking the system to launch the next one.

Service

A service doesn't have a visual user interface, but rather runs in the background for an indefinite period of time. For example, a service might play background music as the user attends to other matters, or it might fetch data over the network or calculate something and provide the result to activities that need it.

Broadcast Receiver

A broadcast receiver is a component that does nothing but receives and reacts to broadcast announcements, often issued by the operating system. Many broadcasts originate in system code ; for example, announcements that the time zone has changed, that the battery is low, that a picture has been taken, or that the user changed a language preference. Applications can also initiate broadcasts ; for example, to let other applications know that some data has been downloaded to the device and is available for them to use.

Content Provider

A content provider makes a specific set of the application's data available to other applications. The data can be stored in the file system, in a SQLite database, or in any other manner that makes sense.

2.1.2 System's architecture

The operating system is composed by five major layers : Applications, Application Framework, Libraries, Android Runtime and Linux Kernel. The present work will mainly affect the Application layer (video player application) and Libraries layer (with the video codec libraries and secure storage). Figure 2.2 shows the Android stack with its different layers.

Figure 2.2 and its description are based on the official documentation. (See [6]).

Applications

Android includes a set of applications like SMS, Contacts, Calendar, etc. Depending if the Android version is delivered by Google or not (Android Open Source Project, See [5]), Google services will be included or not (GMail, GTalk,etc).

Application Framework

By providing a software development kit (SDK), Android offers developers the ability to build extremely rich and innovative applications. They are free to take advantage of the device hardware such as access location information, run background services, set alarms, add notifications to the status bar, etc.

The application architecture is designed to simplify the reuse of components ; any application can publish its capabilities and any other application may then make use

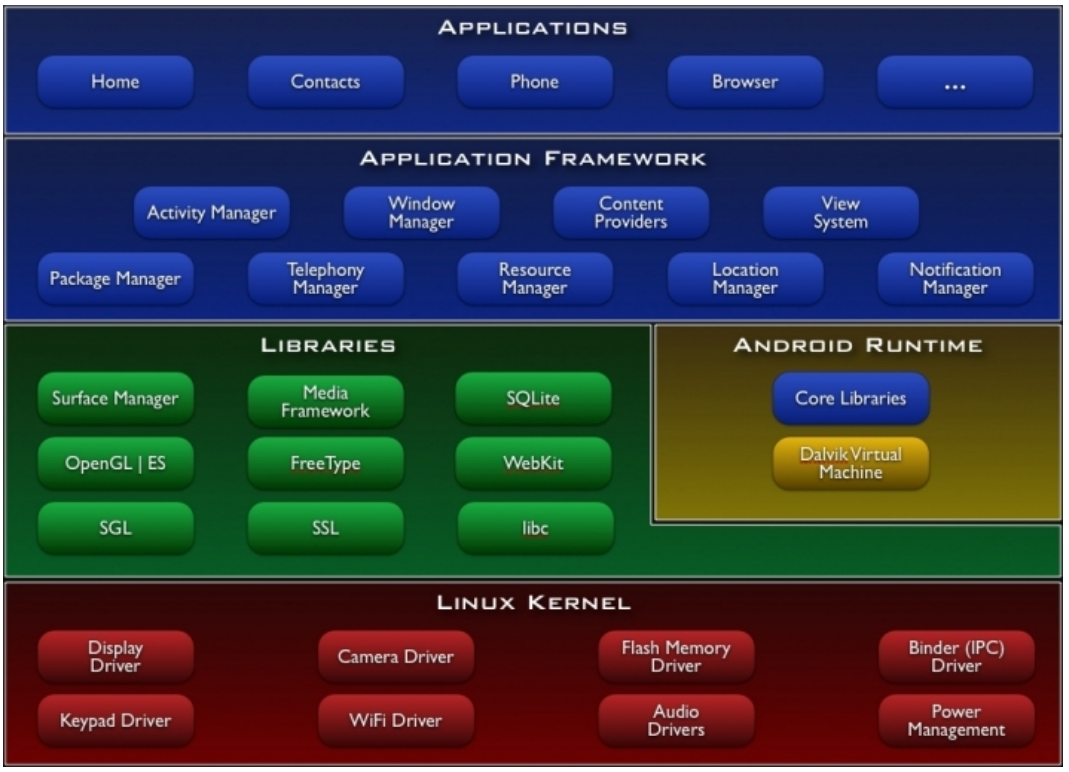

FIGURE 2.2 – Android's software stack

of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user (like the SMS Manager, Browser, Dialer,...).

Android Runtime

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM runs classes compiled by a Java language compiler that have been transformed into the .dex format.

The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

Libraries

Android includes a set of C/C++ libraries used by various components of the Android system. Some of these capabilities are exposed to developers through the Android application framework. Some of the core libraries are : System C library, Media Libraries, 3D libraries, SQLite, etc.

Kernel

Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

2.1.3 Security model

Signing applications

Before any deployment, a developer has to sign its application because the operating system uses the certificate as a means of identifying the author of an application and establishing trust between applications. The certificate is not used to control which application a user can install. The certificate does not need to be signed by a certificate authority : it is perfectly allowable, and typical, for Android applications to use self-signed certificates.

During development, the SDK and the Integrated Development Environment (IDE) Eclipse will compile your application and signed them with a "Debug" certificate. This certificate can be used for development purposes on an emulator, included into the SDK. Any application signed with the "Debug" certificate will be refused by the operating system on a device. The SDK provides the necessary tools to signed and optimize a compiled application.

Using Permissions

A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user. This includes reading or writing the user's private data (such as contacts or e-mails), reading or writing another application's files, performing network access, keeping the device awake, etc.

While installing an application on a device, the system will assign it a `UserId` (This name is really confusing. Android considers the application as a "user" in the Unix sense. "AppId" or something similar would have been more clear) This value is unique and remains constant for the duration of the applications life on that device.

Every application is running in a sandbox (one for each `UserId`), created by the virtual machine (Dalvik Virtual Machine). If any interaction with components outside the sandbox is needed, some explicit permission needs to be requested before (at installation time) on the application's manifest. This mechanism grants an application the right to use third party elements such as software (Agenda, Contact...) or hardware (Camera, GPS, Network...).

If the application tries to use anything under permission without declaring it, an exception will rise. For example, if the application tries to access the Internet without asking the permission, the system will refuse to open the necessary socket and rise an

UnknownHostException.

Listing 2.1 shows a demo application's Manifest where permission to access the Internet and to write on the external storage (sd-card) are requested.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="be.webperso.android.asplayer"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity (...) >
    </application>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```

Listing 2.1 – Using Permission with the AndroidManifest file

Declaring Permissions

Sometimes, declaring some application's permission may be useful. It means that the application opens its interfaces to third party applications and control the way it is done.

Listing 2.2 shows how an application that wants to control who can start one of its activities could declare a permission for this operation. It declares a name of the permission to refer to it. It also provides a description for the end user, permission group in order to understand what risks may be in stake and a protection level.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="be.webperso.android.asplayer"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity (...) >
    </application>
    <permission android:name="be.webperso.android.asplayer.permission.
        DEADLY_COSTLY_ACTIVITY"
        android:label="@string/perm_label_deadlyCostlyActivity"
        android:description="@string/perm_desc_deadlyActivity"
        android:permissionGroup="android.permission-group.COST_MONEY"
        android:protectionLevel="dangerous" />
</manifest>
```

Listing 2.2 – Declaring Permission with the AndroidManifest file

Permissions can be defined on the following software components.

- **Activity** : Activity permissions restrict who can start the associated activity. The permission is checked during the instantiation of the targeted activity. If the caller does not have the required permission, an exception (SecurityException) is thrown.
- **Service** : Service permissions restrict who can start or bind the associated service. The permission is checked during the instantiation or the binding of the targeted service. If the caller does not have the required permission, an exception (SecurityException) is thrown.

- **Broadcast Receiver** : Broadcast Receiver permissions restrict who can send broadcast to the associated receiver. The permission is checked after the broadcast has been send, as the system tries to deliver the submitted broadcast to the given receiver. As a result, a permission failure will not result in an exception being thrown back to the caller ; it will just not deliver the intent.
- **Content Provider** : Content Provider permission restricts who can access data in a Content Provider. The permissions are checked when you first retrieve a provider (if you don't have either permission, a SecurityException will be thrown), and as you perform operations on the provider.

Protection level and User Experience

When declaring permission, a protection level has to be defined. Its purpose is to understand how "dangerous" this feature might be to warn the user. It is interesting to see that Google may decide to change this politics in the next releases of Android depending on how this security model successfully prevents attacks or not.

The following permission level exists.

- **Normal** : "*A lower-risk permission that gives an application access to isolated application-level features, with minimal risk to other applications, the system, or the user. The system automatically grants this type of permission to a requesting application at installation, without asking for the user's explicit approval*"
- **Dangerous** : "*A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user. Because this type of permission introduces potential risk, the system may not automatically grant it to the requesting application. (...)*"
- **Signature** : "*A permission that the system is to grant only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval.*"
- **SignatureOrSystem** : "*A permission that the system is to grant only to packages in the Android system image or that are signed with the same certificates. Please avoid using this option, (...)*"

Content Providers and URI permissions

Sometimes, very precise permissions are needed. For example, a Content Provider may want to protect itself with read and write permissions, while its direct clients also need to hand specific URIs to other applications for them to operate on. A typical example is attachments in a mail application. Access to the mail should be protected by permissions, since this is sensitive user data. However, if a URI to an image attachment is given to an image viewer, that image viewer will not have permission to open the attachment since it has no reason to hold a permission to access all e-mail.

2.2 Digital Right Management

Digital Right Management (DRM) enables Content Providers to grant and control permissions for media objects that define how they should be consumed. The DRM system is independent of the media object formats and the given operating system or run-time environment. The media objects controlled by the DRM can be a variety of things : games, ring tones, photos, music clips, video clips, streaming media, etc. A content provider can grant appropriate permissions to the user for each of these media objects. The content may be distributed with cryptographic protection ; hence, the Protected Content is not usable without the associated Rights Object on a Device (See sample on Appendix B). Given this fact, fundamentally, the users are purchasing permissions embodied in Rights Objects and the Rights Objects need to be handled in a secure and un-compromising manner.

The present work will be based on the Open Mobile Alliance Digital Right Management Standard version 1.0 (See [21]), version 2.0 (See [22]) and version 2.1 (See [23]).

2.2.1 General principles

The OMA DRM Standard defines three main methods to deliver DRM content : Forward Lock, Combined Delivery and Separate Delivery. Figure 2.3 from the OMA DRM 1.0 Specification document (See [18])shows the basic principles of every methods.

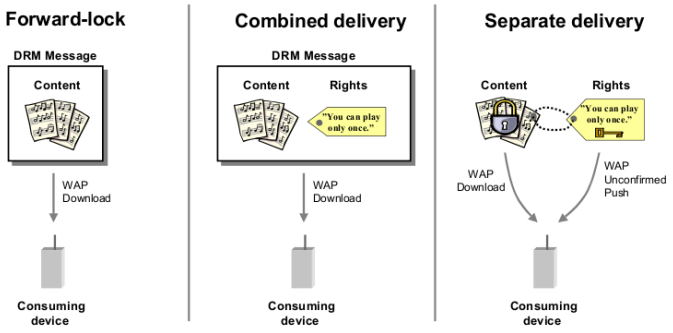

FIGURE 2.3 – DRM Delivery Methods

Forward Lock

Forward Lock is frequently used on "classic" mobile phones (GSM's) and can effectively prevent illegal copying of files. In Forward Lock mode, the content is packaged and sent to the mobile terminal as a DRM message (See Figure 2.4). Then the mobile terminal can use the content, but cannot forward it to other devices or modify it. In most handsets, the Forward Lock content is not encrypted when it is received or when stored in phone memory.

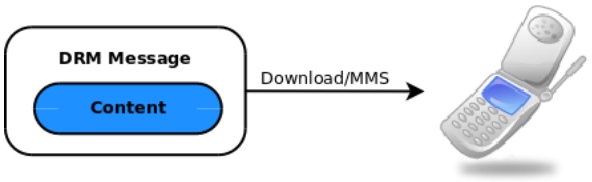

FIGURE 2.4 – Forward Lock Mode

Combined Delivery

Combined Delivery is an extension of Forward Lock. In Combined Delivery mode, the digital rights are packaged with a content object in the DRM message (See Figure 2.5). The user could use the content as defined in the rights object, but could not forward or modify it.
Rights Objects are used to specify consumption rules for DRM content. The Rights Expression Language (REL) defined by OMA DRM (defined as a mobile profile of ODRL v1.1. [17]) specifies the syntax, based on XML, and semantics of permissions and constraints governing the usage of DRM Content. [11]

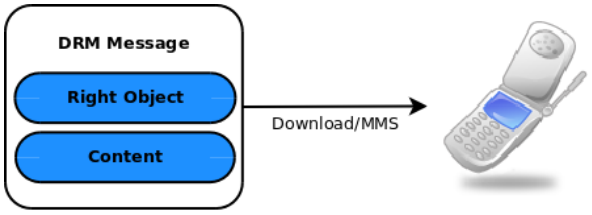

FIGURE 2.5 – Combined Delivery Mode

Separate Delivery

In the Separate Delivery mode, the content and rights are packaged and delivered separately (See Figure 2.5). The content is encrypted into DRM Content Format (DCF) using a symmetric cryptography method and can be transferred in an unsafe way such as Bluetooth, IrDA and via Email. The Rights Object and the Content Encryption Key (CEK) are packaged and transferred in a safe way such as an unconfirmed Wireless Application Protocol (WAP) push. The terminal is allowed to forward the content message but not the associated Rights Object.

FIGURE 2.6 – Separate Delivery Mode

Superdistribution is a Separate Delivery application which encourages digital content being transferred freely and is typically distributed over public channels. But the content recipient has to contact the retailer to get the Rights object and CEK to use or preview the content. It is illustrated on Figure 2.7.

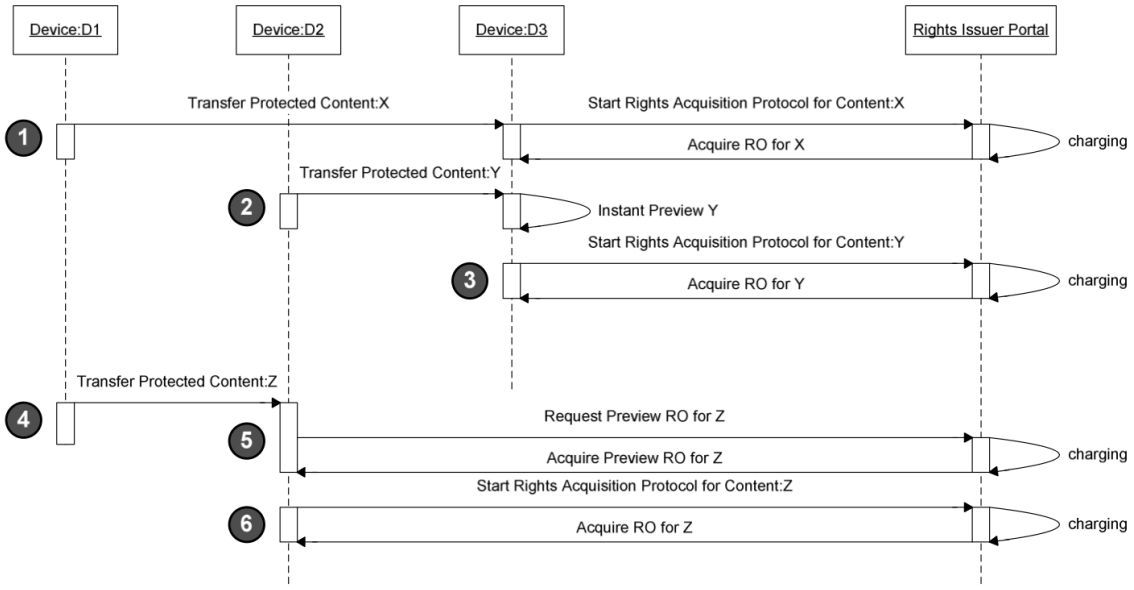

FIGURE 2.7 – Super Distribution

2.2.2 Standard differences

The first version of the protocol (OMA DRM 1.0) was designed on the assumption that the mobile terminal is reliable. In the Forward-lock mode and the Combined Delivery mode, the content is not encrypted. In the Separate Delivery mode, the symmetric encryption key is not encrypted. The media content can be stolen if the mobile terminal is hacked or the Right Object message with the CEK is revealed.

Then, OMA DRM 2.0 was released in order to support more application scenarios like preview, download, MMS, streaming media, unconnected device, etc. It was a more reliable and flexible way to ensure copyright. Unfortunately, most handsets in the market do not support this version of the standard.

OMA DRM 2.1 has been developed as a result of market feedback. The main differences between OMA DRM v2.0 and OMA DRM v2.1 are the addition of several features on top of OMA DRM v2.0, such as metering, content differentiation, Right Object installation confirmation, etc.

Every new version of the standard introduce new security mechanisms, such as the usage of a Public Key Infrastructure (introduced in OMA DRM 2.0).

2.2.3 DRM content distribution overview

The first step for DRM content distribution is to package the content in a secure content container (DCF). The DRM content is encrypted with a symmetric Content Encryption Key (CEK). With OMA DRM, content can be pre-packaged, meaning that it would not be necessary to do the package on the fly. (See Figure 2.8, Step 1.)

As all DRM Agent have a unique private/public key pair and a certificate containing additional informations such as manufacturer, device type, software version, etc. It allows the Content and Right Issuers to securely authenticate a DRM Agent. (See Figure 2.8, Step 2.)

When the DRM Agent asks a Rights Objects for a media content, it receives an XML document, expressing the permissions and constraints associated with the content. The Rights Objects also contains the CEK to ensure that DRM content cannot be used without an associated Rights Object. (See Figure 2.8, Step 3.)

Before delivering the Rights Objects, sensitive parts are encrypted (e.g. the CEK). It results in a Rights Object bounded to a particular DRM Agent. This ensures that only this target DRM Agent can access the Right Object and thus the DRM Content. (See Figure 2.8, Step 4.)

Finally, the Rights Object and the associated DCF can now be delivered to the target DRM Agent. As both objects are secure, they can be delivered using any transport mechanisms. They can be delivered together (Combined Delivery) or separately (Separate Delivery). (See Figure 2.8, Step 5a and 5b.)

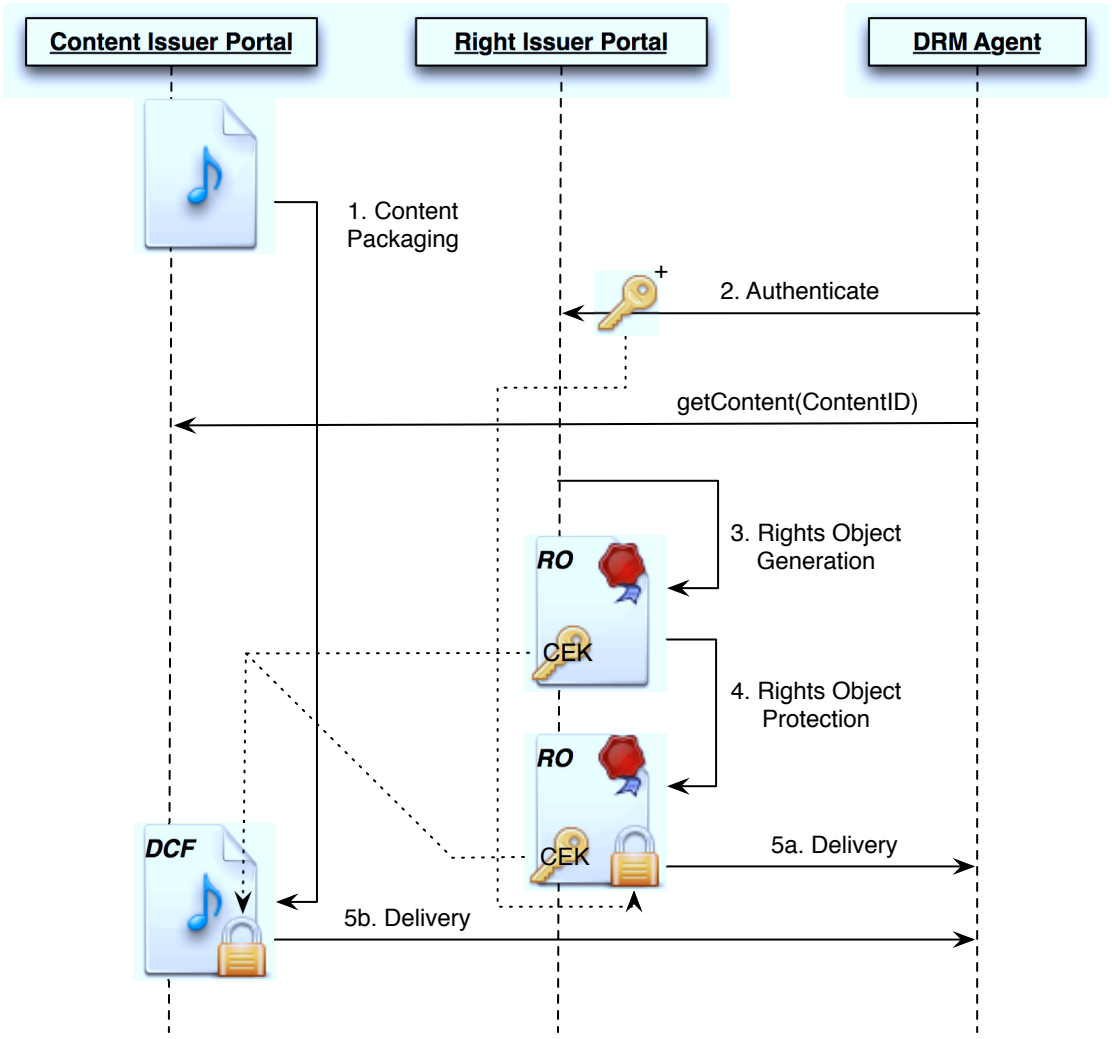

FIGURE 2.8 – DRM Content delivery overview

2.2.4 Dealing with DRM protected content

When the DRM Agent, located on the mobile device, receives the encrypted media content, it starts with parsing the metadata of the file (the file structure is base on the ISO Base Media File Format). Listing 2.3 from the OMA DRM 2.1 Specification (See [13]) shows some of the header boxes provided by a OMA DRM 2.1. encrypted media content.

```
aligned(8) class OMADRMCommonHeaders extends FullBox('ohdr', version,
   0) {
        // Encryption method
        unsigned int(8) EncryptionMethod;
        // Padding type
        unsigned int(8) PaddingScheme;
        // Plaintext content length in bytes
        unsigned int(64)        PlaintextLength;
        // Length of ContentID field in bytes
        unsigned int(16)        ContentIDLength;
        // Rights Issuer URL field length in bytes
        unsigned int(16)        RightsIssuerURLLength;
        // Length of the TextualHeaders array in bytes
        unsigned int(16)        TextualHeadersLength;
        // Content ID string
        char    ContentID[];
        // Rights Issuer URL string
        char    RightsIssuerURL[];
        // Additional headers as Name:Value pairs
        string  TextualHeaders[];
        // Extended headers boxes
        Box     ExtendedHeaders[];
}
```

Listing 2.3 – OMA DRM 2.1 Common Headers Box

With all of those information parsed, the DRM Agent has then the possibility to check if it already has a RO associated with the *Content ID* of the file. It checks then if such file exists. If the RO doesn't exist or the user hasn't the right to do the current action, the DRM Agent will contact the Rights Issuer (RI) to get a correct one. When one RO is received, the DRM Agent has to explicitly confirm the RO installation on the secure storage.

In order to contact the RI, any OMA 2.1 compliant DRM Agent must follow a protocol called "ROAP" (Right Object Acquisition Protocol). This protocol defines every interaction possible between a Right Issuer and a DRM Agent. In the RI side, the execution of ROAP may involve one or more OCSP (Online Certificate Status Protocol) responders, but this interactions are not always mandatory. The following interactions are possible :

- *The 4-pass Registration Protocol* : This protocol is only executed at first contact in order to exchange security information. It can also be executed when there is a need to update security information or when the DRM Time is deemed inaccurate by the RI. When this protocol is successful, it results in the establishment of

an RI Context in the device, containing RI specific security informations (like its certificate which must be saved).

- *The 2-pass Identification Protocol* : This protocol allows the RI and the DRM Agent to exchange credendials. It does not include any mutual authentication or integrity protection. Then, DRM IDs should be verified at a later point in time using one of the variant of the protocol.
- *The 2-pass and the 1-pass Rights Object Acquisition Protocol* : This protocol is used in order to get Rights Objects by the DRM Agent. This protocol assumes the DRM Agent has a pre-established RI Context with the RI. The 1-Pass Rights Object Acquisition Protocol is initiated unilaterally by the RI in order to meet the messaging/push use case.
- *The 4-pass and the 3-pass Confirmed Rights Object Acquisition Protocol* : If the RI requires to confirm the delivered RO's installation, this protocol is used. Both protocols are exactly the same as the 2-pass and 1-pass Rights Object Acquisition Protocol, but extended with two additional messages confirming the RO installation.
- Some few other protocols are defined, but are not used in this work : *the 2-pass Join Domain Protocol*, *the 2-pass Leave Domain Protocol*, *the 2-pass Metering Report Protocol* and *the 2-pass Rights Object Upload Protocol*.

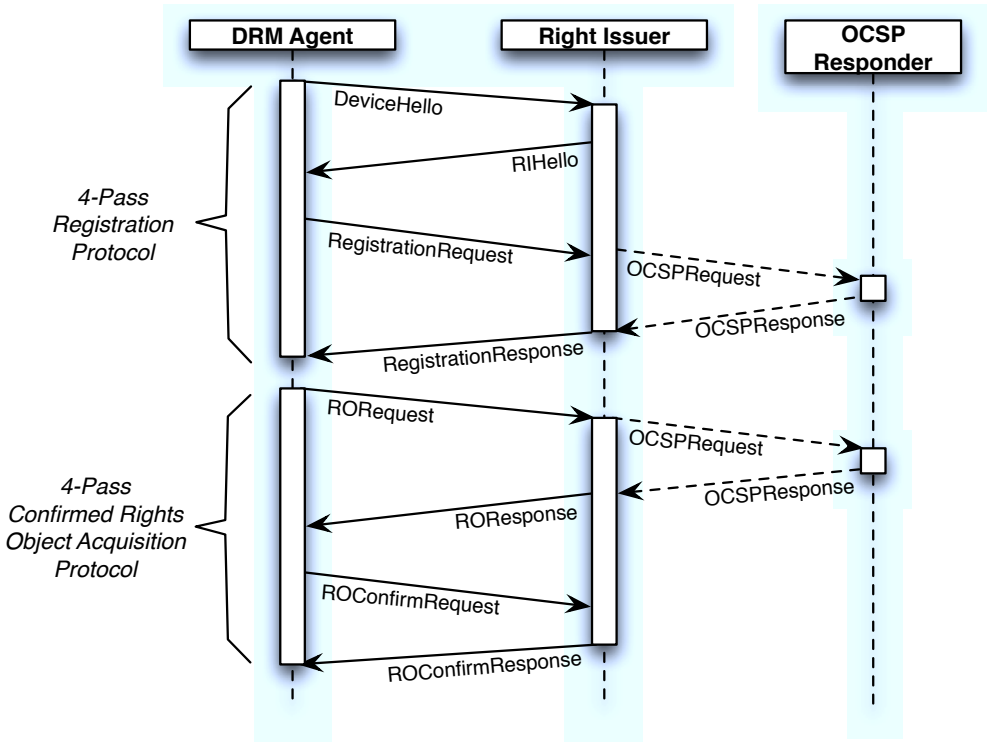

FIGURE 2.9 – Example of ROAP exchanges

Figure 2.9 shows a typical RO aquisition with ROAP. Samples of every message can be found in [17].

2.3 Video Technology

Nowadays, media content on the web is delivered using three main delivery methods : traditional streaming, progressive download and adaptive streaming.

The present description of the adaptive streaming technology is based on the Microsoft Smooth Streaming Specification.(see [8])

2.3.1 Traditional Streaming

Real Time Streaming Protocol (RTSP), developed by the IETF in 1998 as RFC2326, is a good example of a traditional streaming protocol. This is a *statefull* protocol i.e. it means that the server keeps tracks of the client's state from its first connection until its disconnection. The client communicates with the server with commands like *PLAY*, *PAUSE* or *TEARDOWN*. (The last one is used to disconnect from the server)

After a session between a client and a server has been established, the server begins sending the media as a steady stream of small packets (the format of those packet may be then RTP or RDT). A typical packet's size is 1452 bytes, which means that in a video stream encoded at 1 megabits per second (Mbps), each packet carries approximately 11 milliseconds of video. RTP packets can be transmitted with either UDP or TCP ; the latter is preferred when firewalls or proxies block UDP packets, but can also lead to increased latency (TCP packets are re-sent until received).

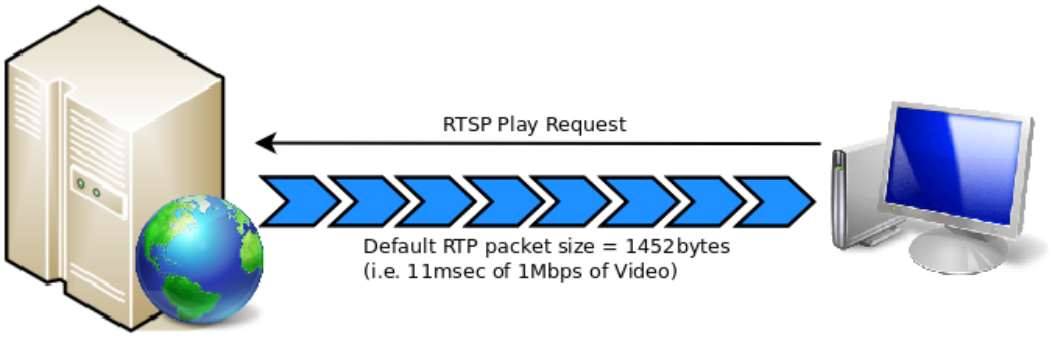

FIGURE 2.10 – RTSP is an example of a traditionnal streaming protocol

2.3.2 Progressive Download

Another common form of media delivery on the Web today is progressive download, which is nothing more than a simple file download from an HTTP Web server. Progressive download is supported by most media players and platforms, including Adobe Flash, Silverlight, and Windows Media Player. The term "progressive" stems from the fact that most player clients allow the media file to be played back while the download is still in progress—before the entire file has been fully written to disk (typically to the Web browser cache). Clients that support the HTTP 1.1 specification can also seek to positions in the media file that haven't been downloaded yet by performing byte range

requests to the Web server (assuming that it also supports HTTP 1.1).

Unlike streaming servers that rarely send more than 10 seconds of media data to the client at a time, HTTP Web servers keep the data flowing until the download is complete. If you pause a progressively downloaded video at the beginning of playback and then wait, the entire video will eventually have downloaded to your browser cache, allowing you to smoothly play the whole video without any hiccups. There is a downside to this behavior as well—if 30 seconds into a fully downloaded 10 minute video, you decide that you don't like it and quit the video, both you and your content provider have just wasted 9 minutes and 30 seconds worth of bandwidth and data.

2.3.3 Http-based Adaptive Streaming

Adaptive streaming is a hybrid delivery method that acts like streaming but is based on HTTP progressive download. It's an advanced concept that uses HTTP rather than a new protocol. This technology may use different codecs, formats, and encryption schemes, it relies on HTTP as the transport protocol and performs the media download as a long series of very small progressive downloads, rather than one big progressive download.

In a typical adaptive streaming implementation, the video/audio source is cut into many short segments ("chunks") and encoded to the desired delivery format. Chunks are typically 2-to-4-seconds long, this length is not too short nor too long (which would increase the overhead of handling the transactions). At the video codec level, this typically means that each chunk is cut along video GOP (Group of Pictures) boundaries (each chunk starts with a key frame) and has no dependencies on past or future chunks/GOPs. This allows each chunk to later be decoded independently of other chunks. The encoded chunks are hosted on a HTTP Web server. A client requests the chunks from the Web server in a linear fashion and downloads them using plain HTTP progressive download. As the chunks are downloaded to the client, the client plays back the sequence of chunks in linear order. Because the chunks are carefully encoded without any gaps or overlaps between them, the chunks play back as a seamless video.

The "adaptive" part of the solution comes into play when the video/audio source is encoded at multiple bitrates, generating multiple chunks of various sizes for each 2-to-4-seconds of video. The client can now choose between chunks of different sizes. Because Web servers usually deliver data as fast as network bandwidth allows, the client can easily estimate user bandwidth and decide to download larger or smaller chunks ahead of time. The size of the playback/download buffer is fully customizable.

Adaptive streaming, like other forms of HTTP delivery, offers the following advantages over traditional streaming to the content distributor :

- It's cheaper to deploy because adaptive streaming can use generic HTTP caches/-proxies and doesn't require specialized servers at each node.
- It offers better scalability and reach, reducing "last mile" issues because it can dynamically adapt to inferior network conditions as it gets closer to the user's

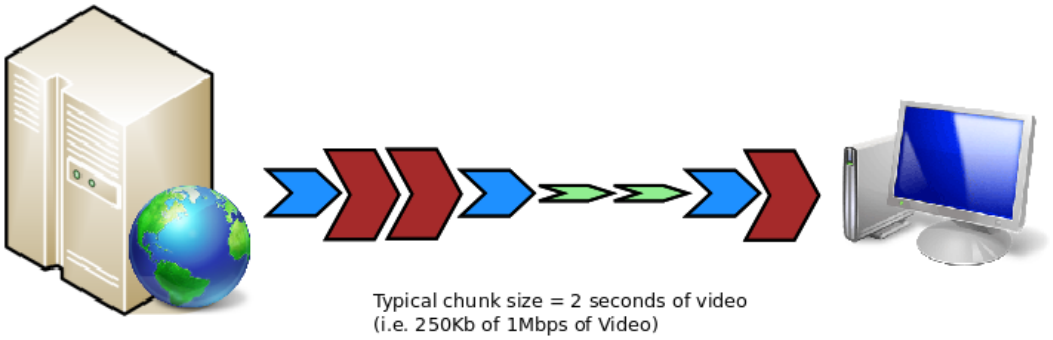

FIGURE 2.11 – Adaptive streaming is a hybrid media delivery method

home.
- It lets the audience adapt to the content, rather than requiring content providers to guess which bit rates are most likely to be accessible to their audience.

It also offers the following benefits for the user :
- Fast start-up and seek times because start-up/seeking can be initiated on the lowest bit rate before moving to a higher bit rate.
- No buffering, no disconnects, no playback stutter (as long as the user meets the minimum bit rate requirement).
- Seamless bit rate switching based on network conditions and CPU capabilities.
- A generally consistent, smooth playback experience.

On the server : "Disk File Format"

The MP4 specification allows various ways to organize data and metadata boxes within a file. Most of the time, it may be useful to have the metadata written before the data so that any client application can have more information about the media content it's about to work with before it effectively play. The MP4 ISO Base Media File Format specification is designed to allow MP4 boxes to be organized in a fragmented manner, where the file can be written "as you go" as a series of short metadata/data box pairs, rather than one long metadata/data pair. The adaptive streaming technology heavily leverages this aspect of the MP4 file specification.

Figure 2.12 shows a file which starts with file-level metadata (moov) that describes the file, but the bulk of the playload is actually contained in the fragment boxes that also carries accurate fragment-level metadata (moof) and the media content (mdat). This figures shows only two fragments, but usually, there is a fragment for each 2 seconds of video or audio.

Fragment to deliver to the client : Wire File Format

When a client requests a video chunk, the server seeks to the appropriate starting fragment in the file and then lifts the fragment out of the file and send it over the wire to the client. Figure 2.13 shows what such file looks like.

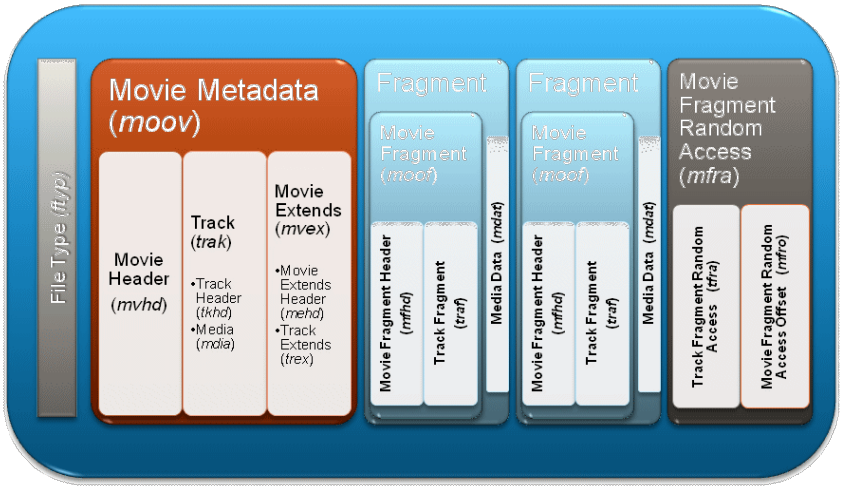

FIGURE 2.12 – Adaptive Streaming File Format (ISMV or ISMA file)

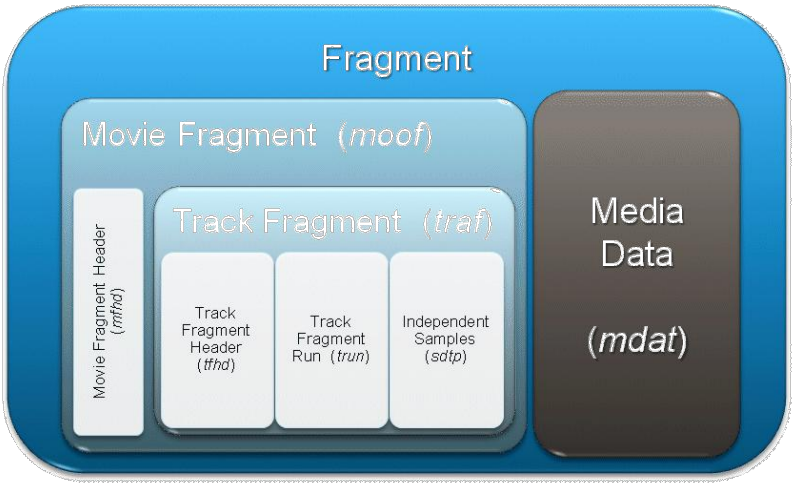

FIGURE 2.13 – Adaptive Streaming Wire Format

Other assets

A typical adaptive streaming architecture contains the following files :

- ***.ismv** This file is the media file itself defined as Disk File Format. There is one of them for each encoded video bitrate.
- ***.isma** This file contains only audio. Sometimes, the audio track can be muxed into an ISMV file instead of a separate ISMA file.
- ***.ism** This file is called "the server manifest". It describes the relationships between the media tracks, bitrates and files on the disk. This file is used by the server, not the client.
- ***.ismc** This file is called "the client manifest". Is describes the available streams to the client ; the codecs used, bitrates encoded, video resolutions, etc. It is the first file to be delivered to the client. A sample is available in Appendix C

2.4 Security

This chapter is based on [9], [29] and [2].

2.4.1 Symmetric and Asymmetric Key Cryptography

Cryptography techniques allow a sender to disguise data so that an intruder can gain no information from the intercepted data. Of course, the receiver (which can be the sender himself) must be able to recover the original data from the disguised data. Encryption enables techniques to identify and authenticate entities (persons or devices).

All cryptography algorithms involve substituting one thing for another, here, taking a piece of plaintext and then computing and substituting the appropriate ciphertext to create the encrypted message.

In symmetric cryptography, the same key is used to encrypt and decrypt (hence the word "symmetric", the same on both sides) ; if you use any other key to decrypt, the result is gibberish. The main issue with this approach is to deal with key management and distribution (as if the key is intercepted by a third party person it becomes then useless). But with asymmetric cryptography (see Figure 2.14 from [2]), the key that's used to encrypt the data does not decrypt it ; only its partner does (hence the word "asymmetric", each side has a different key).

Different algorithms exists using symmetric or asymmetric cryptography. Here, Advanced Encryption Standard (AES) will be used on the secure storage and the content encryption, for symmetric encryption with a key of 128 to 256 bits.

2.4.2 Public Key Infrastructure

Public-key cryptography offers not only a powerful mechanism for encryption but also a way to identify and authenticate other individuals and devices. Before this technology can be used effectively, however, their is one drawback to deal with. Just as with symmetric-key cryptography, key management and distribution are an issue with public-key cryptography.

At the end-user and relying parties side, they have to provide their public key to third party entities. But the problem is that if that public key is susceptible to manipulations while transit. If an unknown person can substitute its public key for the valid one, the attacker could forge digital signatures and allow encrypted messages to be disclosed to unintended parties. That's why it's crucial to assure users that the key they receive is authentic and that it came from the intended party.

But if some trusted entities exist, this problem can be solved easily. A user can send his public key within a document called an "X.509 Certificate", which declares that key is effectively associated with that particular user and that trusted party certifies it. Those

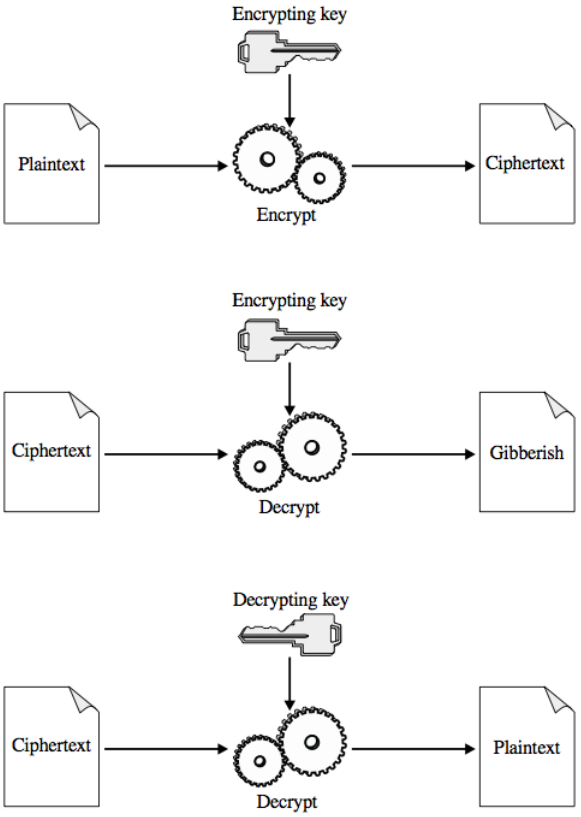

FIGURE 2.14 – In asymmetric crypto, the encrypting key cannot be used to decrypt ; you must use its partner

trusted entities are called "Certification Authorities" (CA). All of those assets and processes compose a Public Key Infrastructure(PKI).

That security mechanism will be used with the ROAP protocol to certify data exchanges.

Chapter 3

Protected content delivering on Android : Solution design

3.1 Product description : Use Case scenarios

3.1.1 Diagram

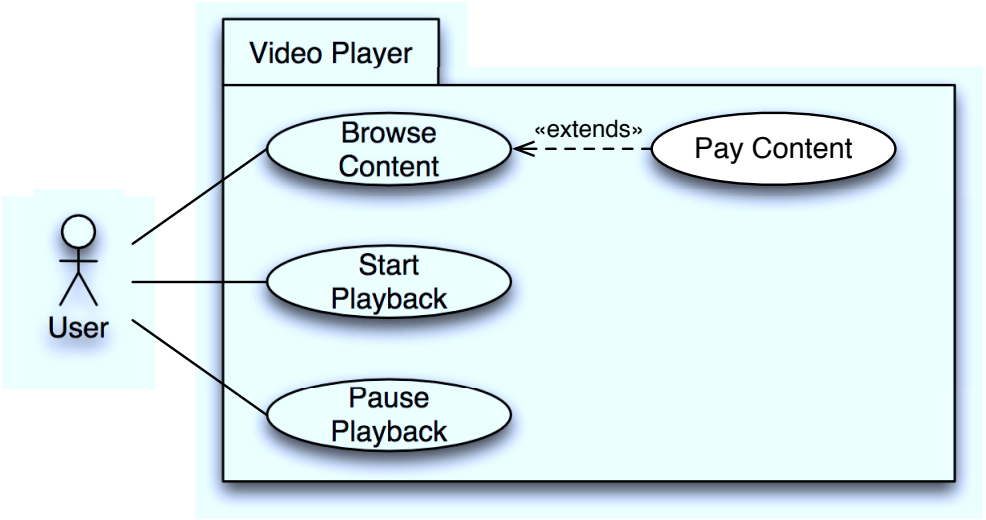

FIGURE 3.1 – Use Case defined for this prototype

3.1.2 Scenario details

Use Case 1

Use Case Name : Browse Content
Summary : The user will browse the content available on the server and choose one.
Pre-condition(s) :

– The user has a viable internet connection.
– The server is online and ready.

Basic course of events :

1. The user starts the prototype and gets connected to the server.
2. He browses through the available content.
3. He chooses a content.

Alternative path(s) : If the chosen content has to be paid, the user is redirected to a payment system (See use case "Pay Content").

Post-condition(s) :

– The user has chosen a content and paid it if necessary.
– He is redirected to the media player (See use case "Start Playback").

Use Case 2

Use Case Name : Start Playback
Summary : The user starts the playback of the content he chose.
Pre-condition(s) :

– The user has paid the content if necessary.
– The playback MAY have already been started and paused.
– The user has a viable internet connection.
– The server is online and ready.

Basic course of events :

1. The media player (re-)starts playing the chosen content.

Alternative path(s) : /
Post-condition(s) :

– The content has been played completely.
– The media player returns to the menu where the user can choose a new content (See use case "Browse Content").

Use Case 3

Use Case Name : Pause Playback
Summary : The user pauses the playback.
Pre-condition(s) :

– A content is playing.

Basic course of events :

1. The media player pauses the current playback.

Alternative path(s) :
Post-condition(s) :

– The content's playback is paused.
– The playback can be re-started (See use case "Start Playback").

Use Case 4

Use Case Name : Pay Content

Summary : Once a content is chosen, if it is not free, the user has to pay for it to play.

Pre-condition(s) :

- The user has chosen a paid content.
- Any payment mechanism is available and ready.

Basic course of events :

1. The payment methods are beyond the scope of this work and are left to the implementer.

Alternative path(s) : /

Post-condition(s) :

- The content is paid.
- The user is redirected to the media player in order to start playback (See use case "Start Playback").

3.2 Design and operating constraints

3.2.1 CMLA security policy

In order to use DRM mechanisms, key pairs and certificates are required in order to encrypt data and sign requests. Content Management License Administrator (CMLA) is an entity formed to provide commercial licenses for the OMA DRM 2.0 (and upper) interoperability specification (See [4]). It defines a "*Client Adopter Robustness Rules*" which consists of a set of good practices and data/attributes to keep secure on the DRM Agent in order to maintain certificates validity (See [3], page 67).

> *(...) Participating Product Implementations shall be manufactured in a manner that is clearly designed to (a) effectively frustrate attempts to discover or reveal Device Private Keys and other confidential values as described in the Confidentiality and Integrity Table in Appendix X and (b) detect unauthorized modifications of values identified as requiring integrity protection in the Confidentiality and Integrity Table in Appendix X and stop the usage of such values if such unauthorized modification is detected.*

Content provider agrees to maintain confidentiality and/or integrity. If not, content provider is exposed to prosecution which can result in fines of up to one million U.S. dollars (US$1,000,000). The Client Adopter Agreement document (See [3]) defines different problematic situations with different fees.

The different data/attribute to secure are defined in Appendix A.

3.2.2 Post-Installable solution and Time to market

In order to reach new customers quickly, the present work will be "post-installable" i.e. an application which can be installed on the device from the market after being purchased. Thus, no security certificates can be considered present at the first launch of the application.

Being "post-installable" presents many advantages like a quicker Time-to-market and easy updates. Indeed, as the application may be installed at any time it must not rely on any system components (which may take some time to be updated). Furthermore, any update can be directly implemented and deployed.

This approach is also platform version independent ; it should work in the same way from Android 1.5. to Android 2.1. and thus targets a lot more customers with the same product and implementation. If the application, at the native layer uses standard code (such as C ANSI), it simplifies the task to port it on other platforms such as iPhone or Blackberry (the core of the application remains the same).

3.2.3 Low computer power

Another huge operating constraint is the low CPU power. In order to play video content smoothly on a mobile device, a video decoder has to be optimized. Furthermore, if the video is protected with OMA DRM, it is often encrypted and then needs more CPU power to be decrypted on the fly.

Table 4.2 shows the evolution of different milestone devices since the first Android release. It clearly shows that it did not evolved in a significant way until begin 2010. Even if new devices runs 1 GHz processors, most of the actual devices on the market are still running 500 MHz processors.

Name :	**Power** :	**Release** :
HTC G1 Dream	528 MHz	October 08
HTC G2 Magic	528 MHz	February 09
HTC Hero	528 MHz	July 09
Motorola Droid/Milestone	550 MHz	December 09
Google Nexus One	1 GHz	January 10

TABLE 3.1 – CPU Power Evolution

3.3 Proposed architecture

Figure 3.2 shows the different components bundled with the proposed architecture of the video player. Every component are detailed as follow.

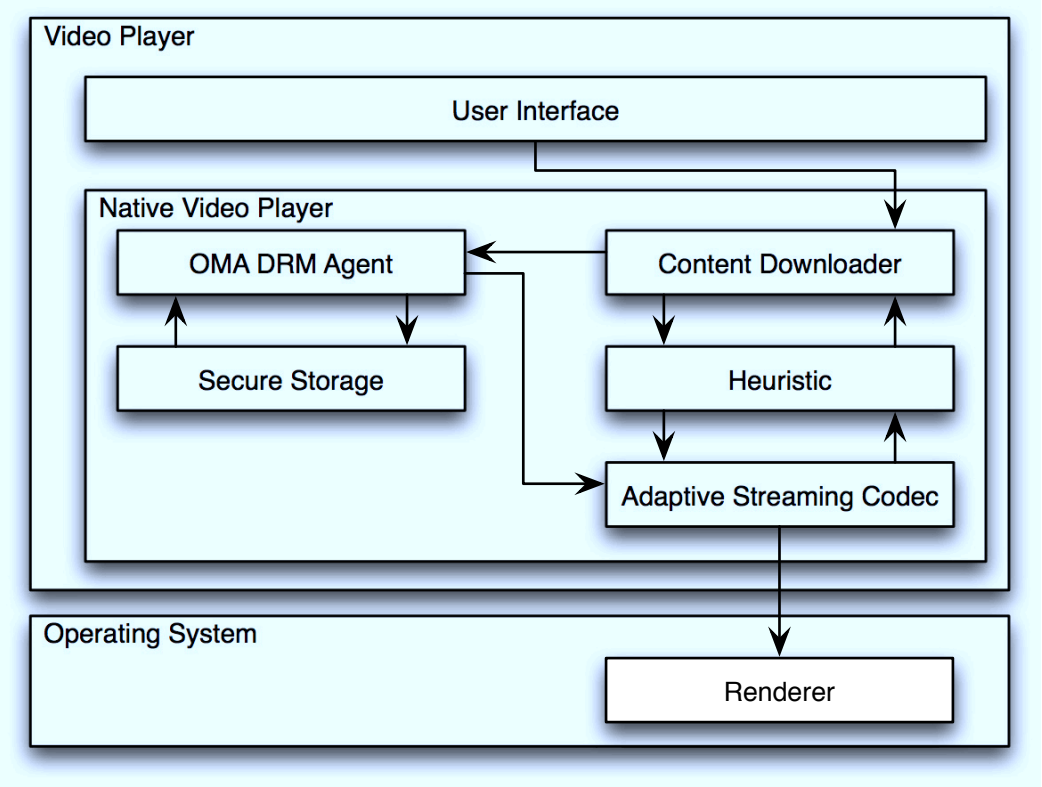

FIGURE 3.2 – Proposed High-Level Architecture

User Interface

This component is a simple user interface to browse the available content, start/pause playback and display the rendered content. It is also used if interactions are required (such as payment).

Content Downloader

This component is used to download the necessary data such as media content (chunks), files (manifest) or Rights Objects. In this work, the protocol used will be HTTP.

OMA DRM Agent

This component is responsible for decrypting protected content following the OMA DRM 2.1 standard. It will be further detailed in section 3.5.

Secure Storage

This component will handle keys and certificates and maintain integrity and confidentiality accordingly to the CMLA Security Policy (See section 3.2.1). It will only be accessible from the embedded OMA DRM Agent for security reasons. It will be further detailed in section 3.6.

Adaptive Streaming Codec

This component will take decrypted media content from the OMA DRM Agent and decode it into frames. Those frames will be rendered using the available operating system features. It will be further detailed in section 3.4.2.

Heuristic

This component is the heart of the adaptive streaming player. It decides if the streaming quality have to be increased, decreased or maintained the same based on data provided by the Content Downloader and the Adaptive Streaming Codec. It will be further detailed in section 3.4.2.

A bundle approach

As most of the components are either not accessible or not existing at all, a bundle approach was chosen. It means that everything the application needs has to be integrated into the application, such as its own video decoder, its own DRM Agent containing its secure storage library.

This approach offers more flexibility and a quicker time to market. As it is not based on system requirements, this application can be installed in a very large set of device and can be updated easily. The major drawback is that the application is heavier to deploy.

3.4 Adaptive Streaming architecture

3.4.1 Playback scenario

With adaptive streaming, the playback starts with the download of a manifest file, specific to the chosen content. It contains the details about the audio and video streams, their quality and chunk details. Once downloaded, the heuristic chooses the minimal quality for both audio and video to start buffering.When playback, the heuristic checks periodically the buffered stream length. If it is less than a threshold (here arbitrarily set to 10 seconds, which may become an application parameter), it adds the next chunk to the downloader. It also checks an heuristic's generated value to adapt stream quality.

Before being ready to be rendered, the chunk passes through different processes. The first one is of course the download process which query the adaptive streaming server, cut it on the server-side and retrieve it. Figure 3.3 shows that process for protected and unprotected content.

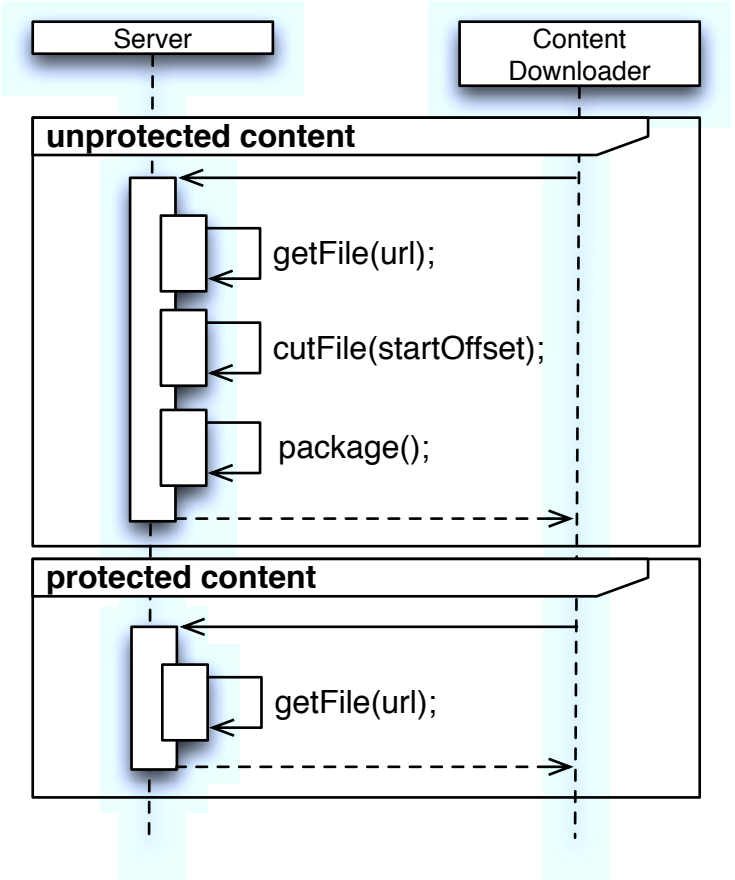

FIGURE 3.3 – File processing at server side

Once downloaded, the chunk has to be checked if it is DRM protected. If it is, this process starts decrypting it with its associated Rights Object. Section 3.5 will detail the

process to get it. When the media content is decrypted, it has to be decoded into frames using the pool of codec. Finally, the system takes the decoded frames to render them.

All of those processes are asynchronous to handle the different processing time. Figure 3.4 shows all the interactions between the previously described components for handling the video part (the same works in parallel for audio and must be synchronised).

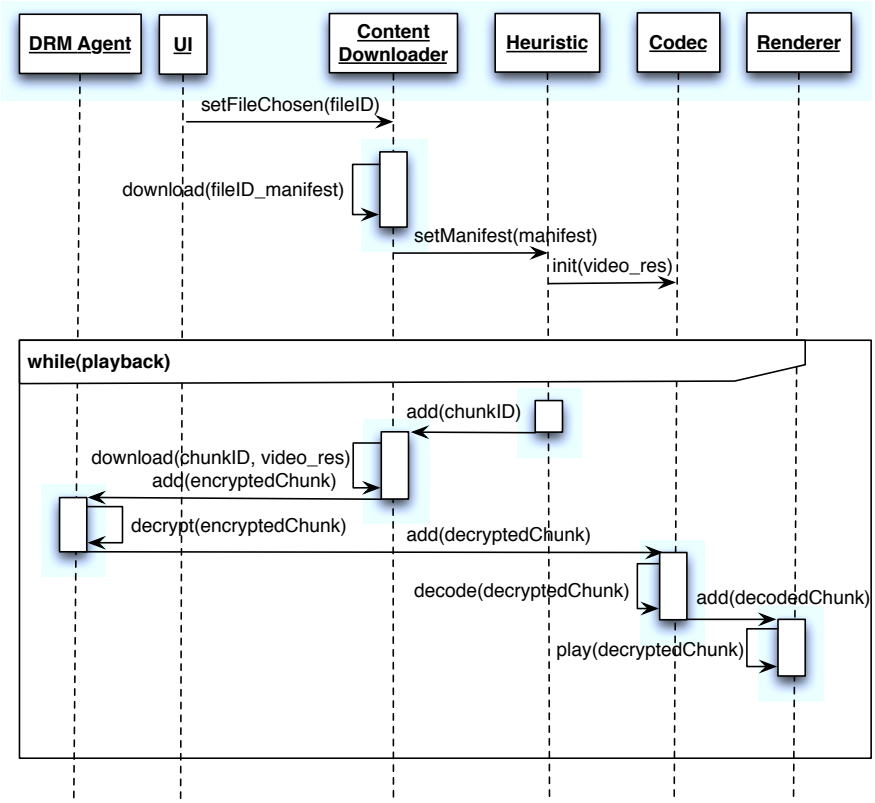

FIGURE 3.4 – Video playback scenario

3.4.2 The video player's core components

Heuristic

As adaptive streaming's most interesting feature is its capability to switch video quality, a heuristic has to be implemented in order to detect if the next video chunk will have the same resolution, a lower one or a higher one.

This component estimates if the video quality has to be changed based on data such as the time required to download the previous video chunk and the estimated time required by the DRM Agent to decrypt a chunk. This component's goal is to maintain a maximum video quality and maximum buffer ready on the queue to be rendered, with the best resolution possible.

It is also important to emphasize that with the current implementation of an adaptive streaming server, defined by Microsoft ([28]), there is only one file per resolution. It means that every time a chunk is asked, it has to be cut form the original complete file and properly packed into the wire format described earlier. This operation occurs for every chunk (a chunk is often 2 to 4 seconds long) and results in a small overhead at the server side which has to be taken into account into the heuristic.

In order to understand exactly how the heuristic works, let's define a couple of variables, related to the different processes used :

- *playbackTime* : this is the playback time of a video or audio chunk, usually 2 to 4 seconds.
- *downloadTime* : this is the time required to download the chunk (this includes the HTTP request, the file's cut on the server and the effective download time)
- *decryptTime* : this is the time required by the DRM Agent to decrypt a given chunk. It does not include the time required by the ROAP protocol if it has to be used (to get the RO for example).
- *decodeTime* : this is the time required by the codec to decode the chunk i.e. extracting and decoding the data which will defined as frames to be rendered.

Those variables are used to calculate a ratio, called α which is used to determine if the quality has to be adapted. α is defined as the ratio between the amount of time required to get a chunk by its time length.

$$processingTime = downloadTime + decryptTime + decodeTime$$
$$\alpha = processingTime/playbackTime$$

For a given chunk, if its α equals "1", it means that it takes the same amount of time to get the chunk than to play it. Having an α the closest below "1" would be the best case scenario, because it would mean that it is not possible to download, decrypt and decode a larger chunk without risking lags while playback.

The α is associated with a given chunk and is used in the decoded chunk FIFO queue (where the chunks are placed after being downloaded, decrypted and decoded). For this queue, an average α is calculated, called α_{dec}. This average value prevents from punctual accidents and shows how the global system behave.

If α_{dec} is bigger than "1", it means that it takes longer to process a chunk than to play it. This is a dangerous situation because it would consume the decoded buffer faster than it can be refilled and then cause lags while playback. The heuristic switches then the quality one step lower to get a lower α_{dec}.

When α_{dec} is below a specific threshold (for example 0.5), the heuristic decides to switch the quality one step higher to try to get a better quality.

With an α_{dec} between "1" and the threshold, the heuristic does not change the quality, because the risk to start processing a too heavy chunk becomes too high.

Figure 3.5 shows the interactions needed to download video chunks of variable bitrate. The resolution is determined by the implemented heuristic. It also emphasize the overhead at the server when the chunk as to be created each time.

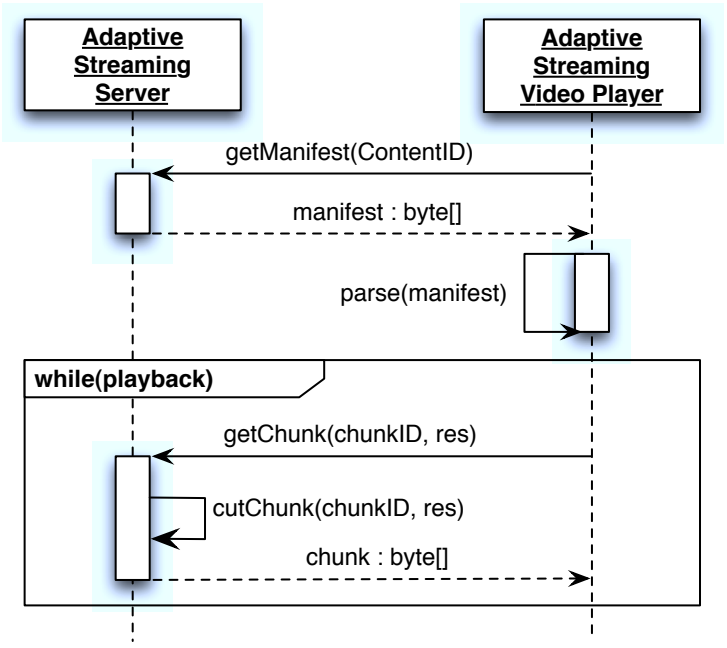

FIGURE 3.5 – Adaptive Streaming download scenario

Codecs

This component receives a decrypted chunk from the OMA DRM Agent and decodes it to produce the frames to be rendered on the display.

Its main purpose is to prepare the decoded frame, encoded with the h.264 technology and contained in a MPEG-4 container. It also has to handle properly the fact that chunks may be encoded with various qualities. The chosen approach for this prototype is simply to create a pool of decoders, one per available quality. Of course, only one of them is really running at a time. The purpose of having a pool of decoders is their availability i.e. if the video quality changes, no decoder has to be initialized, it already is. Of course, it causes a small overhead at the initialization of the playback but offers a quicker response when video quality changes. It is really important regarding the fact that there is not much time to prepare the next chunk, only the time given by the decoded chunks buffer.

Changing video resolution while playback is a brand new feature offered by adaptive streaming. It gives the end user a very smooth user experience while playback, with no video lag at the cost of resolution changes. Furthermore, the present prototype uses HTTP to download video chunks. It becomes then cheaper to deploy that technologies regarding such servers are widespread. The only problem is that the HTTP server has to be capable of slicing the original video file into a chunk properly. To do that, some proprietary and some free solutions [1] exists on the market such as Microsoft's IIS extension [2].

1. http://smoothstreaming.code-shop.com/trac
2. http://www.iis.net/extensions/SmoothStreaming

3.5 Streaming with OMA DRM 2.1

3.5.1 Standard specifications

As this product will consume protected content, let's look at what the OMA DRM 2.1 specification (see [20]) says about streaming media :

> *(...)Thus, the basic concept for the application of OMA DRM to streaming services is that OMA DRM ROs, and the ROAP, are used in the same way as for downloadable objects/DCFs. This is specified in this standard. The exact way of protecting streams, storing streams at a streaming server, and transporting streams to a Device (including associated signaling) are not specified in this specification. It is the responsibility of streaming standardisation bodies to define appropriate mechanisms that work seamlessly together with the concept laid out in the DRM specification, especially with the RO concept and format. (...)*

This can be modeled as in figure 3.6 (from [11]). First, the client browses the content available on the server and chooses one. After any form of payment, he receives a "Streaming Token". Then, with this token, the client has to ask the proper Rights Object related to its token. Finally, with the associated Rights Object, he can initiate a streaming session and will be capable of decrypting and rendering properly the content.

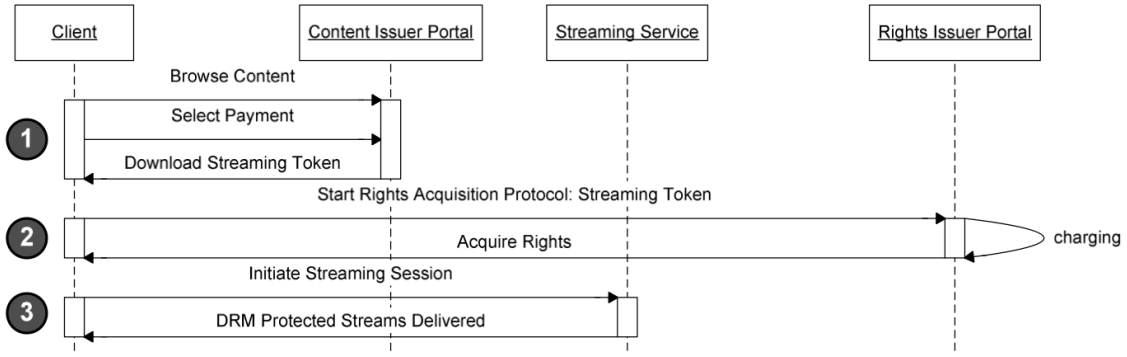

FIGURE 3.6 – Streaming with OMA DRM 2.1

Using the OMA DRM RO's and ROAP in the same way as OMA DRM 2.1 specify implies two main steps :

1. DRM Agent initialization
2. Rights Objects Acquisition

3.5.2 DRM Agent Initialization

With the OMA DRM 2.1 standard, a DRM Agent requires certificates and key pairs to encrypt and sign data. Those security items have to be securely stored (see section 3.6). As it is not possible to embed those elements with the application (because it is the same application that is distributed to any customer via the existing Android Market), the approach was thus to download all of those data at first launch via HTTPS (for security reasons, HTTP cannot be used for sensitive data).

Of course, the necessary previous steps of the ROAP protocol have to be followed as described in [20], such as the 4-pass Registration Protocol.

3.5.3 Rights Object Acquisition

In the approach described by the standard, the Rights Object is retrieved before the content acquisition which means that the DRM Agent knows in advance which Rights Issuer to work with. It is also possible to retrieve directly the protected content, and using its metadata, starting the ROAP protocol to get the RO. It's more flexible, because any RI can be used This second approach will be used in this work.

Here, as the DRM Agent is directly integrated into the video player processes, it will behave as described :

- When it receives an encrypted chunk (video or audio), it reads the metadata to get the Content ID and the Rights Issuer URL.
- With the Content ID, it checks with the secure storage (described in section 3.6) if a Rights Object is available.
  - If it exists, the DRM Agent starts to decrypt the chunk.
  - Else, the DRM Agent start the ROAP protocol to retrieve the Rights Object and save it into the secure storage. (While the ROAP protocol is executed, the video player can fill in the downloaded chunks buffer to save some time).

Figure 3.7 shows this process :

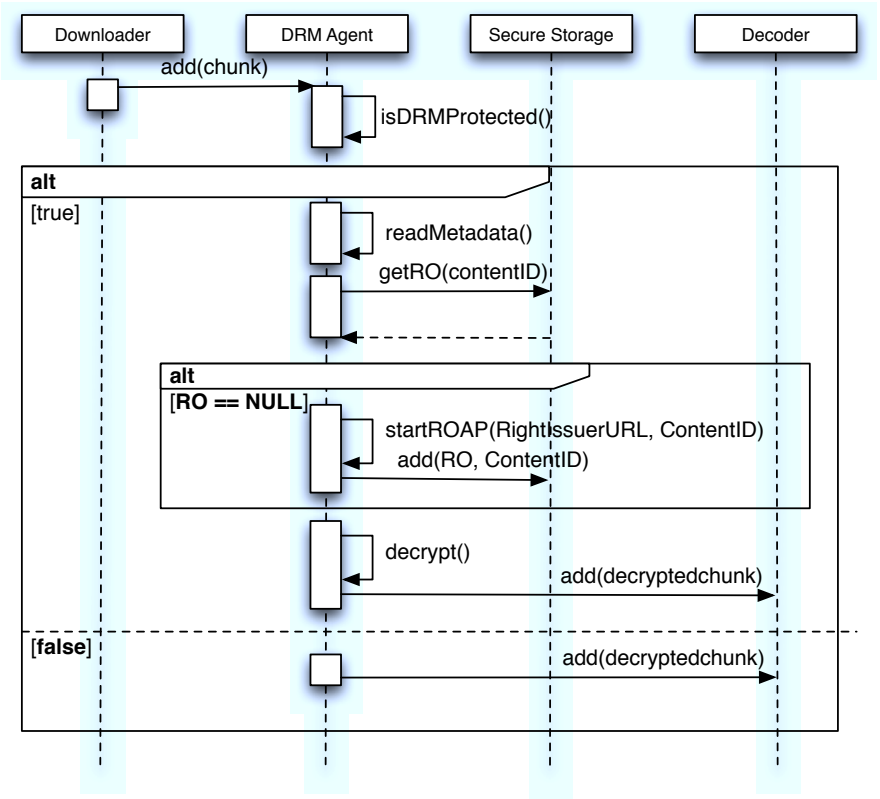

FIGURE 3.7 – DRM Agent Process

3.6 Secure storage : a way to securely store sensitive files

Storing securely Rights Objects and other certificates is a major issue in most DRM environment. Several approaches can be used to mitigate the risk of disclosing sensitive keys.

3.6.1 Trusted Platform Module

The first one is simply to use a dedicated Trusted Platform Module (TPM) (See [27]). TPM is an embedded micro-chip inside the device, where an unauthorized access to such shielded and isolated area would be practically impossible. Only applications considered to belong to a Trusted Software Stack (TSS) would have access to such data by a single entry point per application. Unfortunately, the Android operating system and the devices do not embed such technology.

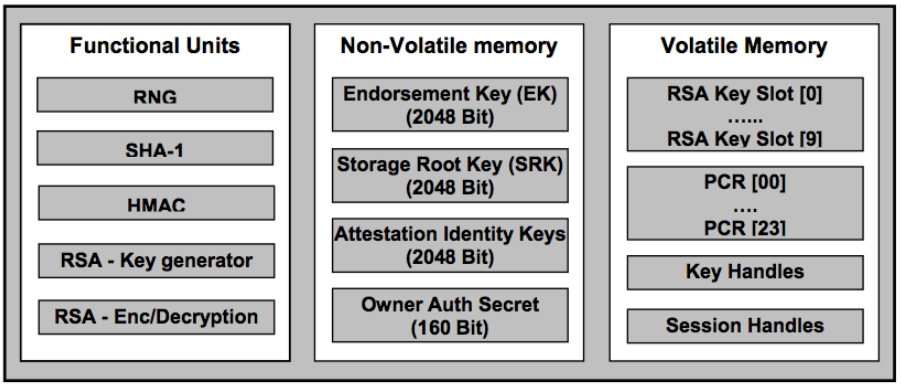

FIGURE 3.8 – Trusted Platform Module Architecture

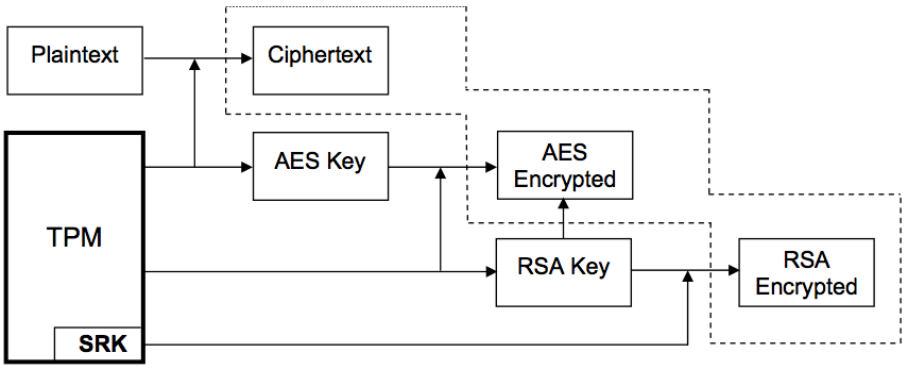

FIGURE 3.9 – Data Encryption using TPM

3.6.2 Encrypted data

As the operating system provides no secured file system nor Trusted Platform Modules, sensitive data would require to be encrypted then stored directly on the application directory. RSA Laboratories designed a protocol, called Public-Key Cryptography Standards #12 (PKCS#12, See [25]) to exchange personal information data, such as keys, certificates, etc. which will set the basements for such secure storage in this work. This standard offers two modes called privacy mode and integrity mode.

The privacy modes are defined by RSA as :

- *Public-key privacy mode* : Personal information is enveloped on the source platform using a trusted encryption public key of a known destination platform. The envelope is opened with the corresponding private key.
- *Password privacy mode* : Personal information is encrypted with a symmetric key derived from a user name and a privacy password. If password integrity mode is used as well, the privacy password and the integrity password may or may not be the same.

The integrity modes are defined by RSA as :

- *Public-key integrity mode* : Integrity is guaranteed through a digital signature on the contents which is produced using the source platform's private signature key. The signature is verified on the destination platform by using the corresponding public key.
- *Password integrity mode* : Integrity is guaranteed through a message authentication code (MAC) derived from a secret integrity password. If password privacy mode is used as well, the privacy password and the integrity password may or may not be the same.

The integrity and privacy modes perfectly meet the security requirements of CMLA as detailed in Appendix A. In this work, unfortunately, the public-key approaches cannot be used in this scope, because it requires a key pair to encrypt data, which must be stored securely, which is the exact problem we are currently trying to solve. PKCS#12 is also implemented on OpenSSL as a software, and provides the tools to create files supported by the eponym standard. Furthermore, OpenSSL is embed into Android, but is no API to PKCS#12 are available to the developpers.

As the public-key approaches are not available for the reasons explained earlier, the password approaches have to be taken. Once more, the question of how to securely store a password is asked. A easy way to mitigate that problem is to generate the password instead of saving it. This password could be a hash of several device or user specific values, such as its own MSISDN, the MEDI, the device version, the operating system version, a hash of a specific part of the memory, etc.

This approach seems to be the best trade-off between security, low computer power,

no secure storage available and no possibilities to modify the operating system.

Data storage scenario

The secure storage will use the API as described in Listing 3.1.

```
#ifndef INCLUDE_SECURE_STORAGE_API_1_1
#define INCLUDE_SECURE_STORAGE_API_1_1

/**
 * -------------------------------------------------------
 * Secure Storage API
 * -------------------------------------------------------
 * @version     1.1
 * @author  Jb Collet
 * @date    04-01-10
 * -------------------------------------------------------
 */

#define FALSE            0
#define TRUE             1

#define MODE_INTEGRITY                  0
#define MODE_INTEGRITY_CONF             1

#define STATUS_OK                                       0
#define STATUS_ATT_NOT_FOUND                    1
#define STATUS_ATT_NOT_MUTABLE                  2
#define STATUS_ID_ALREADY_USED                  3
#define STATUS_UNKNOWN_ERROR                    4
#define STATUS_STORAGE_CORRUPTED                5
#define STATUS_WRONG_PARAMS                     6
#define STATUS_STORAGE_NOT_FOUND                7

#define ITEM_SECRET             0
#define ITEM_KEY                        1
#define ITEM_CERTIFICATE        2
#define ITEM_CRL                        3

/**
 * -------------------------------------------------------
 * This method is used in order to add an attribute into the secure
 * storage
 * -------------------------------------------------------
 * @param attrId
 *      The attribute's identifier to find it back later (with the get
 *      method)
 * @param isMutable
 *      TRUE if the attribute can be modified later (with the edit method), else
    FALSE
 * @param mode
 *      MODE_INTEGRITY if the attribute has to maintain integrity only
 *      MODE_INTEGRITY_CONF if the attribute has to maintain integrity and
    confidentiality
 * @param data
 *      An array of bytes containing the attribute's value to be stored
 * @param item_type
 *      ITEM_SECRET if the attribute is a "secret"
 *      ITEM_KEY if the attribute is a key
 *      ITEM_CERTIFICATE  if the attribute is a certificate
 *      ITEM_CRL if the attribute is a CRL
 * @return
 *      STATUS_OK if the attribute has been stored successfully
 *      STATUS_ID_ALREADY_USED if the attribute's identifier is already used, then
    the attribute is not stored
 *      STATUS_STORAGE_CORRUPTED if the storage has been corrupted, then the
    attribute is not stored
 *      STATUS_WRONG_PARAMS if a wrong parameter has been given to the method
```

```
 *      STATUS_UNKNOWN_ERROR if an unknown error occurred, then the attribute is
     not stored
 * ------------------------------------------------------------
 */
int secure_storage_add (char* attrId, int isMutable, int mode, int item_type, char*
     data);

/**
 * ------------------------------------------------------------
 * This method is used in order to retrieve an attribute from the secure storage
 * ------------------------------------------------------------
 * @param attrId
 *      The attribute's identifier to find it back later (with the get method)
 * @param data
 *      An array of bytes containing the attribute's value to be stored
 * @return
 *      STATUS_OK if the attribute has been retrieved successfully
 *      STATUS_ATT_NOT_FOUND if the attribute is not found on the secure storage
 *      STATUS_STORAGE_NOT_FOUND if the secure storage file has not been found
 *      STATUS_STORAGE_CORRUPTED if the storage has been corrupted, then the
     attribute is not retrieved
 *      STATUS_UNKNOWN_ERROR if an unknown error occurred, then the attribute is
     not retrieved
 * ------------------------------------------------------------
 */
int secure_storage_get (char* attrId, char* data);

/**
 * ------------------------------------------------------------
 * This method is used in order to edit an attribute on the secure storage
 * ------------------------------------------------------------
 * @param attrId
 *      The attribute's identifier to find it back later (with the get method)
 * @param data
 *      An array of bytes containing the attribute's value to be stored
 * @return
 *      STATUS_OK if the attribute has been edited successfully
 *      STATUS_ATT_NOT_FOUND if the attribute is not found on the secure storage
 *      STATUS_ATT_NOT_MUTABLE if the attribute has been stored as a non-editable
     attribute, then the attribute is not edited
 *      STATUS_STORAGE_NOT_FOUND if the secure storage file has not been found
 *      STATUS_STORAGE_CORRUPTED if the storage has been corrupted, then the
     attribute is not edited
 *      STATUS_UNKNOWN_ERROR if an unknown error occurred, then the attribute is
     not edited
 * ------------------------------------------------------------
 */
int secure_storage_edit (char* attrId, char* data);

/**
 * ------------------------------------------------------------
 * This method is used in order to delete an attribute from the secure storage
 * ------------------------------------------------------------
 * @param attrId
 *      The attribute's identifier to find it back later (with the get method)
 * @param data
 *      An array of bytes containing the attribute's value to be stored
 * @return
 *      STATUS_OK if the attribute has been deleted successfully
 *      STATUS_ATT_NOT_FOUND if the attribute is not found on the secure storage
 *      STATUS_STORAGE_NOT_FOUND if the secure storage file has not been found
 *      STATUS_STORAGE_CORRUPTED if the storage has been corrupted, then the
     attribute is not deleted
 *      STATUS_UNKNOWN_ERROR if an unknown error occurred, then the attribute is
     not deleted
 * ------------------------------------------------------------
 */
int secure_storage_delete (char* attrId);

/**
```

```
 * ------------------------------------------------------------
 * This method is used in order to get the name of a status
 * ------------------------------------------------------------
 * @param statusNbr
 *       The number of a status received after the execution of a method
 * @return
 *       The human-readable name of a status
 * ------------------------------------------------------------
 */
char* secure_storage_pstatus(int statusNbr);

#endif
```

Listing 3.1 – Secure Storage API

Figure 3.10 demonstrate the internal processes running behind the "Add" method available with the API.

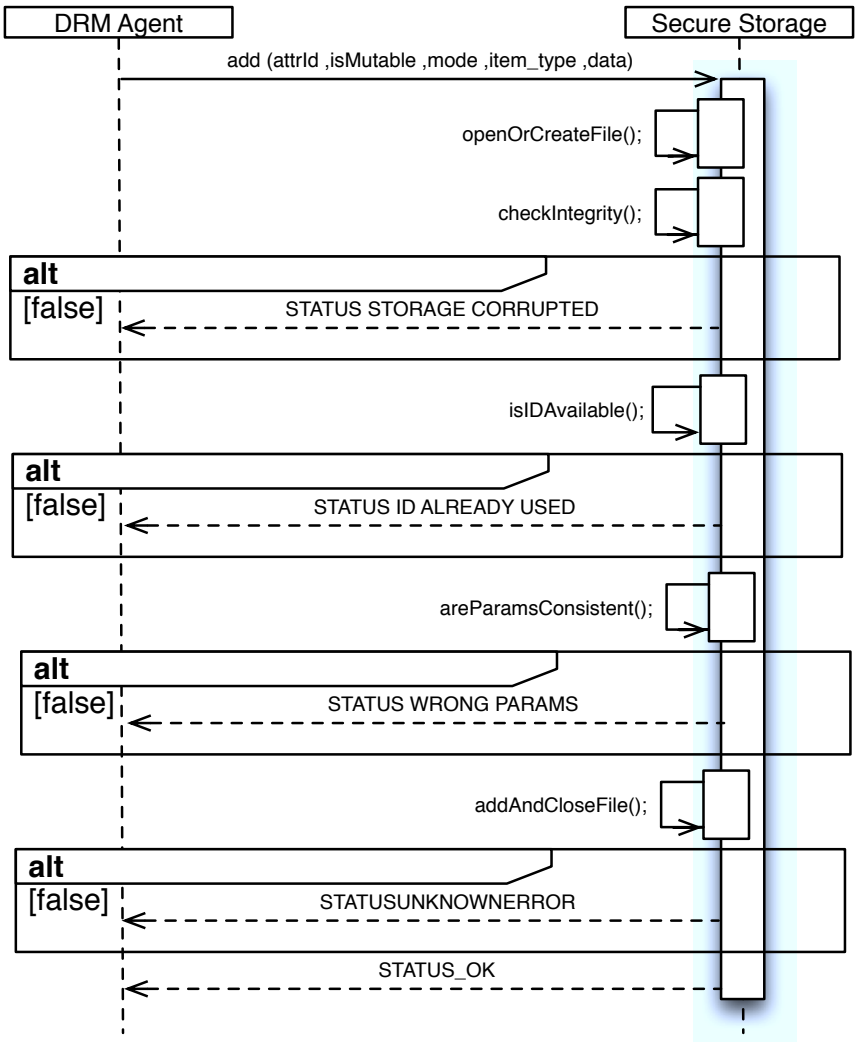

FIGURE 3.10 – Data storage using the secure storage API

Standard adaptation

Given the previously described API and the context restrictions, we can define the secure storage structure as in Figure 3.11. This also includes definitions from PKCS#7 (See [24]).

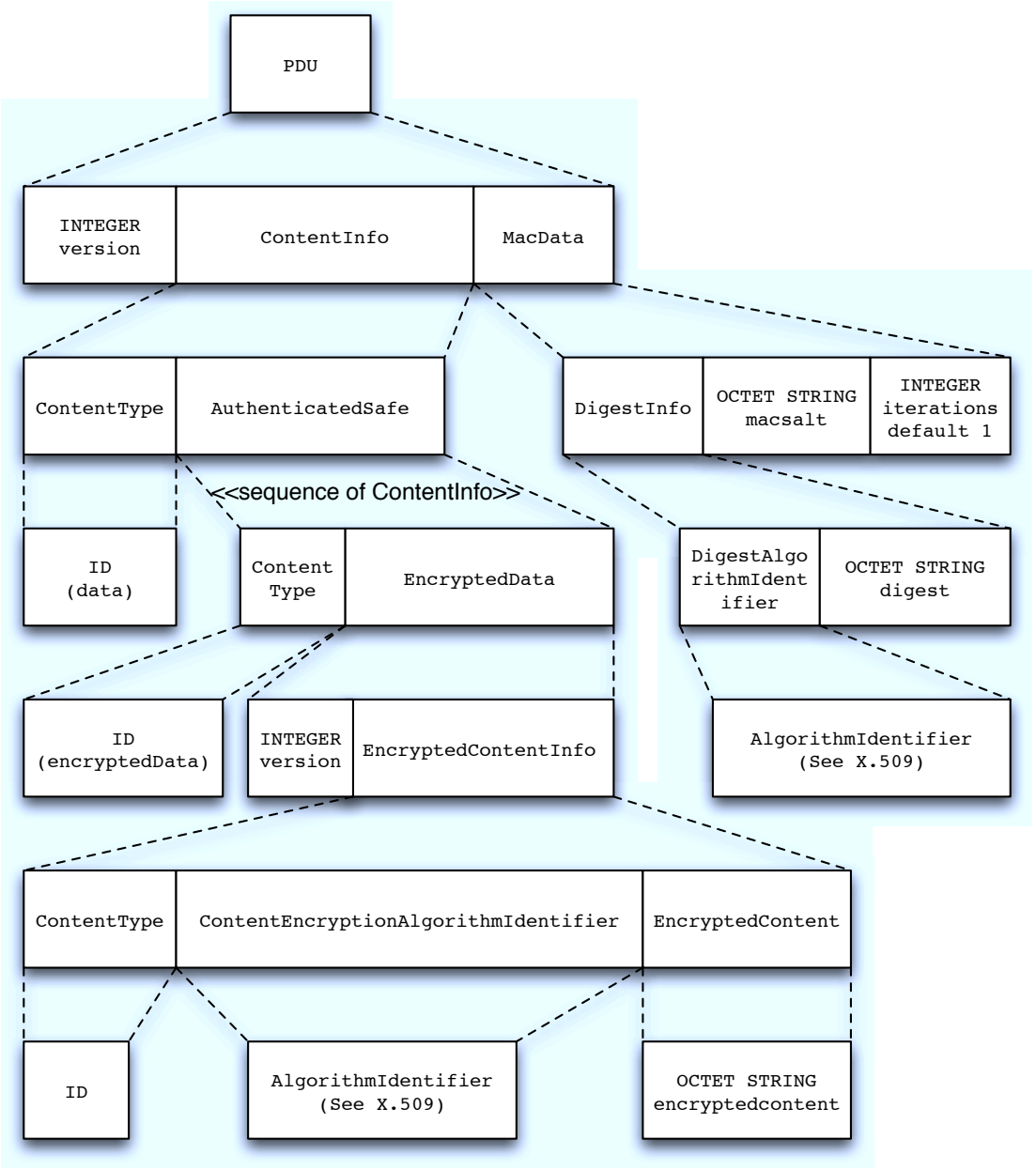

FIGURE 3.11 – Secure Storage structure, based on PKCS#12

Here, the header of the file, the Protocol Data Unit (PDU) represents a sequence of bits in machine-independent format constituting the secure storage file. It contains three fields, an integer for the version (mandatory set to "3" with PKCS#12), a ContentInfo box and a MacData.

Here the MacData is mandatory (it is an optional field in PKCS#12) due to the fact that only password integrity and privacy modes are used. This MacData box contains again three boxes : DigestInfo, the macsalt (as a String) and an iteration counter (as an integer). The DigestInfo box contains informations about the digest algorithm used (the AlgorithmIdentifier defined by the X.509 specification) and the digest itself(as a String).

About the ContentInfo, it contains the data. It is composed of two boxes, the ContentType (which contains an identifier related to the content, here "data"), and a box called AuthenticatedSafe.

The AuthenticatedSafe is a sequence of ContentInfo's. Each ContentInfo has a ContentType set as "encryptedData". The second box, called EncryptedData, contains the encrypted content and the encryption information. The EncryptedData contains the EncryptedContentInfo, which describes the content with a ContentType (set according to the content), a ContentEncryptionAlgorithmIdentifier (which identifies an algorithm as defined by the X.509 specification) and the encrypted content itself (as a String).

Chapter 4

Implementation and experiments

The prototype described in the document is composed of different parts : the video player, the server which includes the media content and the DRM Server which delivers Rights Objects related to protected content.

Unfortunately, this prototype does not support audio. It may be supported in further releases. Its main purpose was to demonstrate the feasibility of rendering DRM protected video at variable bitrate on Android.

4.1 The video player

4.1.1 The Android Application

The proof of concept described in this document was implemented using the Android Software Development Kit (SDK) and Native Development Kit (NDK).

The NDK was used to build and compile the core of the application into a shared library, here called "*libasplayer.so*" . It includes the video player and the used third party libraries except TinyXML.

This core library was wrapped into a Java application written with the SDK and used through a Java Native Interface (JNI) glue. It is composed of two activities : *ListActivity.java* and *PlayerActivity.java*.

The first activity showed, *ListingActivity.java* (See Figure 4.1) propose a set of movie to the user with their names, their descriptions (replaced by the url on the screenshot) and their prices. The user has the possibility to choose one media content which triggers the start of the second activity.

The second activity, *PlayerActivity.java* (See Figure 4.2) initiate the adaptive streaming video player. When the player estimates enough data has been buffered (value defined of 3 chunks), the play button is enabled, enabling the user to start the playback. He also has the possibility to press again the play button to pause, or to click on the

fullscreen button to remove the control bar (this control bar comes back when the user clicks somewhere on the screen).

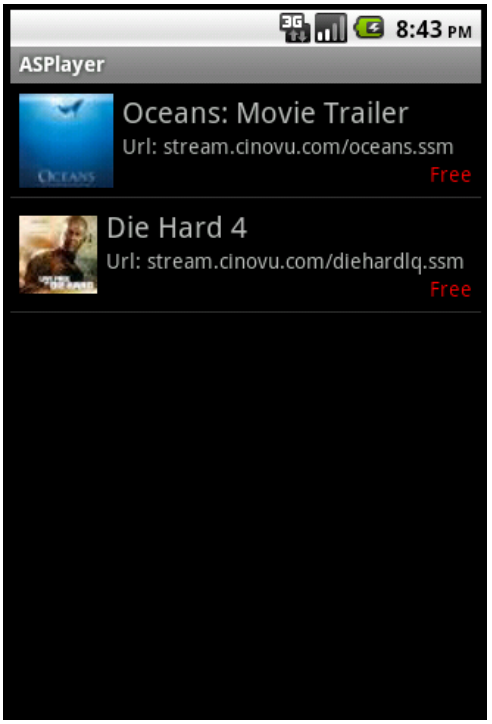

FIGURE 4.1 – ListingActivity (Screenshot from the 13-08-2010)

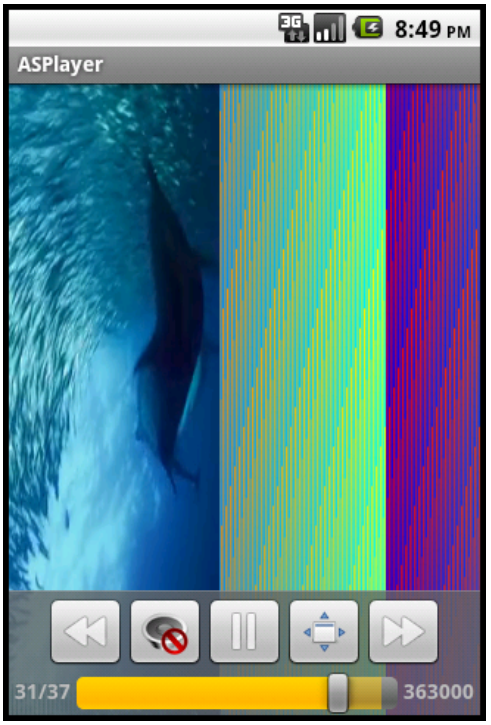

FIGURE 4.2 – PlayerActivity (Screenshot from the 13-08-2010)

The application does not offer any permission for third party software, but uses the following permissions :

- **android.permission.INTERNET** : This permission is required to access the Internet (here to download the available movie list, the manifests and media chunks).

- **android.permission.WRITE_EXTERNAL_STORAGE** : This permission is required to store data on the external SD-Card. It was not a mandatory one, but used only for development purpose (when a chunk does not work, it is stored to be analyzed later).

4.1.2 Implementation

General idea

Figure 4.3 illustrate the general idea behind the prototype, a pipeline architecture.

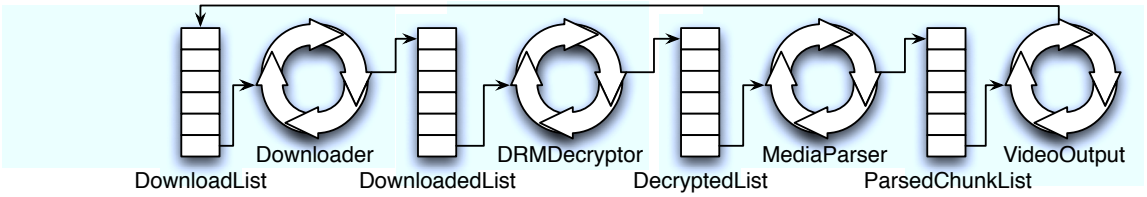

FIGURE 4.3 – General idea Diagram

High Level Class Diagram

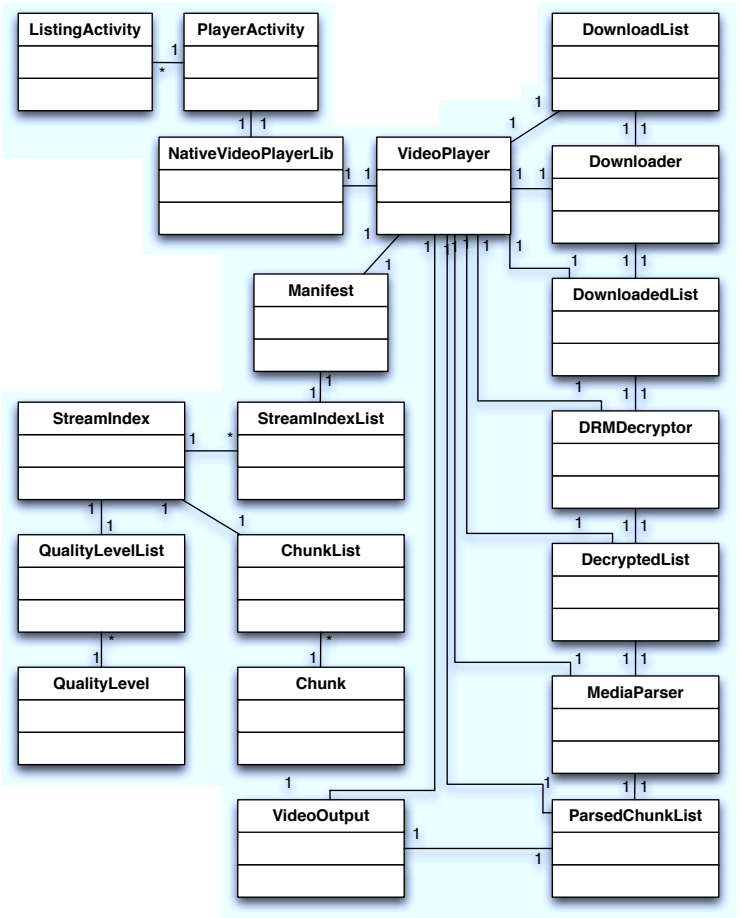

FIGURE 4.4 – High Level Class Diagram

ListingActivity and PlayerActivity

ListingActivity and PlayerActivity are the user interface parts of the application written with the SDK. Those parts are the only ones written in Java. They propose content to the final user and display the playback. They are detailed in section 4.1.1.

NativeVideoPlayerLib

NativeVideoPlayerLib is a JNI file which handles every call from the Java code (here from PlayerActivity). It initiates the video player, forward the different methods such as play/pause. This file and the next described ones are compiled into a shared library called "*libasplayer.so*" by the Android NDK. It also describes the interface which every Java software must use in order to use the compiled library "*libasplayer.so*" (the whole native video player).

Those interfaces are :

```
public static native int native_init(int screen_width, int screen_height, String
    content_url, String content_port, String content_path);
public static native int native_step();
public static native void native_play();
public static native void native_pause();
public static native boolean native_isPlaying();
public static native int native_getDuration();
public static native int native_getCurrentPosition();
public static native int native_getCurrentBitrate();
public static native int native_getBufferPercentage();
public static native void native_seekTo(int i);
public static native void native_setMute(boolean b);
public static native void native_destroy();
```

Listing 4.1 – Java Native Interfaces of libasplayer.so

VideoPlayer

This class is the native adaptive streaming for protected content video player itself. It manages the download, parsing of manifests and data, decryption of protected content, rendering... It is initialized, handled and destroy by the NativeVideoPlayerLib component.

Manifest, StreamIndexList and StreamIndex

While the VideoPlayer is initialized, the content's path is set, resulting in the download by the Downloader of the client manifest. (See Appendix C for a sample of such manifest).

When the manifest is downloaded, it is parsed in order to retrieve information such as the available audio and video streams. The Manifest contains one StreamIndexList which contains every StreamIndex described while parsing the downloaded file. A StreamIndex contains one QualityLevelList and one ChunkList.

QualityLevelList and QualityLevel

A QualityLevelList contains every available stream quality, called QualityLevel, for a given content. A QualityLevel is defined by the following, extracted from the downloaded manifest :

1. a bitrate (ex : 363000 kbps)
2. a fourCC code designing a codec (ex : H264, WVC1)
3. a width (ex : 336 px)
4. a height (ex : 192 px)
5. a codec private data to initiate the codec while rendering (used every time a new chunk is rendered)

Example :

```
<QualityLevel Bitrate="363000" FourCC="H264" Width="336" Height="192"
    CodecPrivateData="000000016764000␣
    DAC2CC505466840000003004000000CA3C50A65800000000168EEB2C8B0" />
```

Listing 4.2 – QualityLevel Sample

ChunkList and Chunk

A ChunkList contains a set of Chunk, which is a fragment of stream. The content is divided in parts of an equal size for every available quality, it enables the possibility of switching between quality without any loss. A chunk is defined by the following, extracted from the downloaded manifest :

1. a chunk identification number (ex : 0, 1, 2, ...)
2. a chunk duration in 10^{-7}s (ex : 41600000)

Example :

```
<c n="1" d="41600000" />
```

Listing 4.3 – Chunk Sample

DownloadList, Downloader and DownloadedList

The Downloader is the component which handles the download of the manifest and the chunks. It uses LibCurl to manage the downloads. It works as a thread which looks into the DownloadList if something has to be downloaded. If so, do the download, calculate the time it took and place the result into the DownloadedList. Else, the thread sleeps for a little while (333333 msec).

DRMDecryptor and DecryptedList

The DRMDecryptor component handles downloaded content from the DownloadList. If the content is DRM protected, it looks into the secure storage to get the key, or retrieve it from the server (via the Downloader). Once the content is decrypted, it is placed into the DecryptedList component. DRMDecryptor also behave as a thread which looks into the DownloadedList and work, or sleep a short time (333333 msec).

MediaParser and ParsedChunkList

Another working thread is the MediaParser component. It takes a content from the DecryptedList, parse it to get the chunks metadata required for the rendering and store it into the ParsedChunkList. Again, if no content is available on the DecryptedList, this thread sleeps for 333333msec.

VideoOutput

The VideoOutput component create an OpenGL context to render properly downloaded content. It takes a content from the ParsedChunkList and for each frame (also called "sample") contained in the chunk, decode it with the FFMpeg library an push it on the buffer which is read by OpenGL as a texture.

Furthermore, every time VideoOutput get a new content from the ParsedChunkList, it start a new thread to complete the download buffer to reach a total of 3chunks.

4.1.3 Third-party libraries bundled with the application

FFMpeg

FFMpeg is used to decode buffered frames. As the downloaded content is encoded with h.264, it was the only decoder embed within the compiled static library. All the other encoders and decoders were removed to get a more lightweight library. This compilation was made with the NDK compiler for ARM (arm-eabi).

Configure command :

```
SDK=$NDK/build/platforms/android-3/arch-arm
TOOLCHAIN=`echo $NDK/build/prebuilt/*/arm-eabi-4.4.0`
export PATH=$TOOLCHAIN/bin:$PATH

EXTRA_CFLAGS="-I$SDK/usr/include -fpic -mthumb-interwork -ffunction-sections -
    funwind-tables -fstack-protector -fno-short-enums -march=armv5te -mtune=xscale -
    -msoft-float"
EXTRA_LDFLAGS="-nostdlib $SDK/usr/lib/libc.so $SDK/usr/lib/libm.so -Wl,-rpath-link=
    $SDK/usr/lib -L$TOOLCHAIN/lib/gcc/arm-eabi/4.4.0"
EXTRA_EXE_LDFLAGS="-Wl,-dynamic-linker,/system/bin/linker $SDK/usr/lib/
    crtbegin_dynamic.o $SDK/usr/lib/crtend_android.o"
EXTRA_LIBS="-lgcc"

FLAGS="--target-os=linux --cross-prefix=arm-eabi- --arch=arm --disable-armvfp"
FLAGS="$FLAGS --prefix=../build/ffmpeg"
FLAGS="$FLAGS --enable-shared"
FLAGS="$FLAGS --enable-small --optimization-flags=-O2"
FLAGS="$FLAGS --disable-encoders --disable-decoders --disable-protocols --disable-
    muxers --disable-demuxers --disable-parsers --disable-devices --disable-filters
    --disable-bsfs"
FLAGS="$FLAGS --enable-decoder=h264"

cd ffmpeg
rm -rf ../build/ffmpeg
mkdir -p ../build/ffmpeg
echo $FLAGS --extra-cflags="$EXTRA_CFLAGS" --extra-ldflags="$EXTRA_LDFLAGS" --extra
    -exe-ldflags="$EXTRA_EXE_LDFLAGS" --extra-libs="$EXTRA_LIBS" > ../build/ffmpeg/
    info.txt
./configure $FLAGS --extra-cflags="$EXTRA_CFLAGS" --extra-ldflags="$EXTRA_LDFLAGS"
    --extra-exe-ldflags="$EXTRA_EXE_LDFLAGS" --extra-libs="$EXTRA_LIBS" | tee ../
    build/ffmpeg/configuration.txt
[ $PIPESTATUS == 0 ] || exit 1
make clean
make -j4 || exit 1
make install || exit 1
```

Listing 4.4 – FFMpeg Configure Command

Once the static libraries available, they have to be defined on the Makefile (Android.mk) in order to be used. The following sample demonstrate how to do it.

```
LOCAL_LDLIBS    := \
        (...)
        ./external_libs/libavcodec.a \
        ./external_libs/libavdevice.a \
        ./external_libs/libavfilter.a \
        ./external_libs/libavformat.a \
        ./external_libs/libavutil.a \
        ./external_libs/libswscale.a \
        ./external_libs/libcurl.a \
        ./external_libs/libtinyxml.a \
        ./external_libs/libgcc.a
```

Listing 4.5 – FFMpeg on the application Makefile

OpenGL

Several ways to render video were studied. The first one was to try to use the existing code used by the operating system on its multimedia framework. This approach's main issue was that this code does not belong to the NDK and thus is not supported. Furthermore, it is strongly discouraged to try to use it, due to its instability through next Android version.

As the NDK natively support OpenGL E.S., this approach was selected and implemented. Unfortunately, this approach is not the more efficient way about memory and CPU usage.

The last possibility, which was not studied deep on this work, is to use the brand new NDK feature about bitmaps. Basically, it is a buffer on which developers can push data which will be rendered.

Using OpenGL E.S. 1.0 on the application is done through this command on the Makefile (it is also possible to use OpenGL E.S. 2.0, but only on Android above version 2.0)

```
LOCAL_LDLIBS    := \
        -lGLESv1_CM \
        (...)
```

Listing 4.6 – OpenGL on the application Makefile

LibCurl

As downloading content from the web is a very well known use case, we used LibCurl in order to do that and handle every exception that might rise.

It was compiled using the Android Source Tree compiler (the operating system sources), but it should also be possible to do it with the NDK. Here, it was then considered as a system static library.

Configure command :

```
export A=/home/jb/sdk/android/mydroid
export CC=$A/prebuilt/linux-x86/toolchain/arm-eabi-4.2.1/bin/arm-eabi-gcc
export SYSROOT=$A/ndk/build/platforms/android-4/arch-arm
export LDFLAGS="-L$SYSROOT/usr/lib -Wl,--gc-sections -nostdlib -lc -lm -ldl -llog
    -lgcc -Wl,--no-undefined,-z,nocopyreloc -Wl,-dynamic-linker,/system/bin/linker
    "
export CFLAGS="-march=armv5te -mtune=xscale -msoft-float -mandroid -fPIC -mthumb-
    interwork -mthumb -mlong-calls -ffunction-sections -fstack-protector -fno-short
    -enums -fomit-frame-pointer -fno-strict-aliasing -finline-limit=64 -
    D__ARM_ARCH_5__ -D__ARM_ARCH_5T__ -D__ARM_ARCH_5E__ -D__ARM_ARCH_5TE__ -
    DANDROID -DOS_ANDROID -D__NEW__ -D__SGI_STL_INTERNAL_PAIR_H -I$SYSROOT/usr/
    include"
./configure --host=arm-eabi --without-zlib --disable-manual
```

Listing 4.7 – LibCurl Configure Command

Usage on the Application Native Makefile (Android.mk)

```
LOCAL_LDLIBS    := \
        (...)
```

```
        ./external_libs/libcurl.a \
```

Listing 4.8 – LibCurl on the application Makefile

TinyXML

TinyXML was used in order to parse downloaded XML document such as the manifest. It was compiled from the Android Source Tree (the operating system sources) instead of the NDK. This library is already existing on the system as a shared library, but for more compatibility, portability and maintainability, it was recompiled as a static library in order to be embeded with the application. (The existing Makefile from the operating system was modified in such way)

The following sample shows how to integrate that library on the application on its Makefile.

```
LOCAL_LDLIBS    := \
        (...)
        ./external_libs/libtinyxml.a \
```

Listing 4.9 – TinyXML on the application Makefile

4.2 The video samples and the server

The video sample used for this prototype is the movie trailer of "*Oceans*" of Jacques Perrin. It was ingested using FFMpeg (not the library, but the software), patched with the video encoder *x.264* from VLC and the audio encoder *faac*.

Given the movie "oceans.mp4", the following command was used to generate a proper MP4 file encoded with h.264 and faac. It starts with a first pass to create "Groups of Pictures" (GOP) in order to be able to switch from one resolution to another without loosing any frame. The second pass encodes the file itself given the constraints defined earlier. The third step consists of creating the ISMV file to be used by the adaptive streaming server. Finally, when every file from every resolution is ready, the manifests (ISMC and ISM files) are created.

Command lines :

```
# This step must be done one time only !
ffmpeg -y -i oceans.mp4 -an -pass 1 -threads 4 -vcodec libx264 -flags +loop+mv4 -
    cmp 256 -partitions +parti4x4+parti8x8+partp4x4+partp8x8+partb8x8 -me_method
    umh -subq 7 -trellis 1 -refs 4 -bf 3 -directpred 1 -b_strategy 1 -mbtree 0 -
    flags2 +bpyramid+wpred+mixed_refs+dct8x8 -coder 1 -me_range 16 -g 100 -
    keyint_min 50 -sc_threshold 40 -i_qfactor 0.71 -qcomp 0.6 -qmin 10 -qmax 51 -
    qdiff 4 -passlogfile pass1-oceans.mp4-oceans.log -r 25 passfile-oceans.mp4

# The following steps must be done for every desired resolution
ffmpeg -y -i oceans.mp4 -an -pass 2 -threads 4 -vcodec libx264 -flags +loop+mv4 -
    cmp 256 -partitions +parti4x4+parti8x8+partp4x4+partp8x8+partb8x8 -me_method
    umh -subq 7 -trellis 1 -refs 4 -bf 3 -directpred 1 -b_strategy 1 -mbtree 0 -
    flags2 +bpyramid+wpred+mixed_refs+dct8x8 -coder 1 -me_range 16 -g 100 -
    keyint_min 50 -sc_threshold 40 -i_qfactor 0.71 -qcomp 0.6 -qmin 10 -qmax 51 -
    qdiff 4 -passlogfile pass1-oceans.mp4-oceans.log -b 79k -r 25 -s 112x64
    oceans_79.mp4

mp4split -o oceans-79k.ismv oceans_79.mp4

(...)

# Those steps create the ISMC and ISM manifests
mp4split -o oceans.ism oceans-384k.ismv oceans-257k.ismv oceans-174k.ismv oceans
    -147k.ismv oceans-79k.ismv
mp4split -o oceans.ismc oceans-384k.ismv oceans-257k.ismv oceans-174k.ismv oceans
    -147k.ismv oceans-79k.ismv
```

Listing 4.10 – Content encoding commands

Available bitrates and resolutions : for the demo content.

Bitrate (kbps) :	**Width** (px) :	**Height** (px) :	**FPS** :
363000	336	192	24
228000	256	144	24
135000	192	112	24
103000	176	96	24
85000	112	64	24

TABLE 4.1 – Demo content available

4.3 Results diagrams

The following section shows the diagrams resulting from the playback of the demo video. A bandwidth drop was simulated before the download of the chunk 16. Figure 4.5 shows the duration for each chunk (which impact on its data size and thus on its process time). Figure 4.6 shows the video quality evolution while playback. Figure 4.7 shows the download time variation, it variates due to the chunk's size and the bandwidth. Figure 4.8 shows the alpha evolution, which impacts on the next chunk's download.

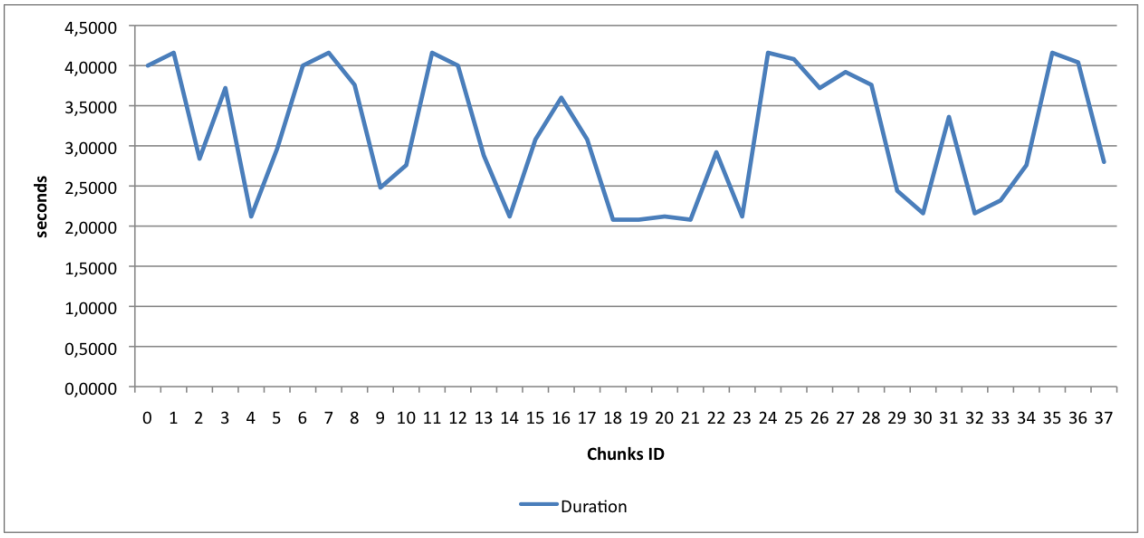

FIGURE 4.5 – Duration of the demo chunks

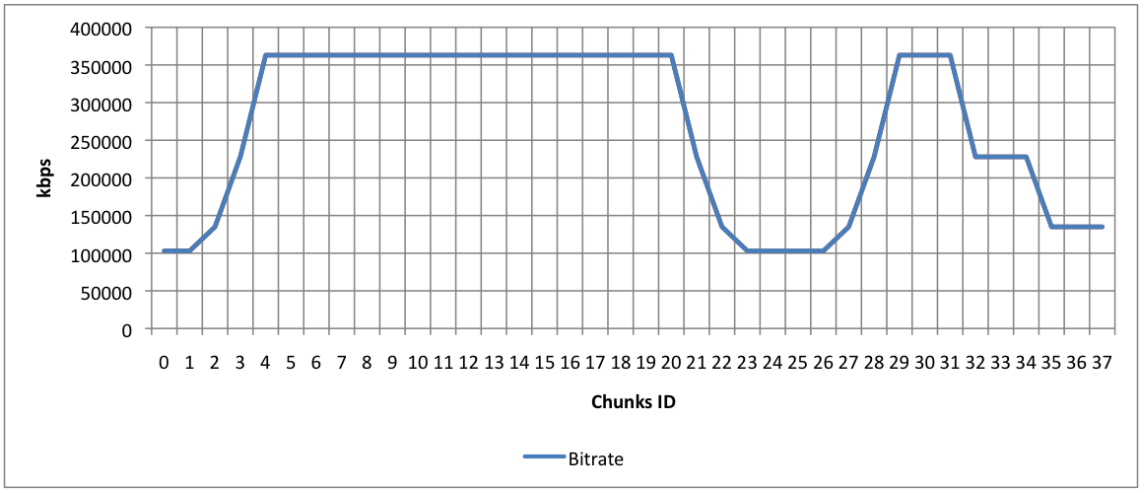

FIGURE 4.6 – Bitrate variation during a demo playback

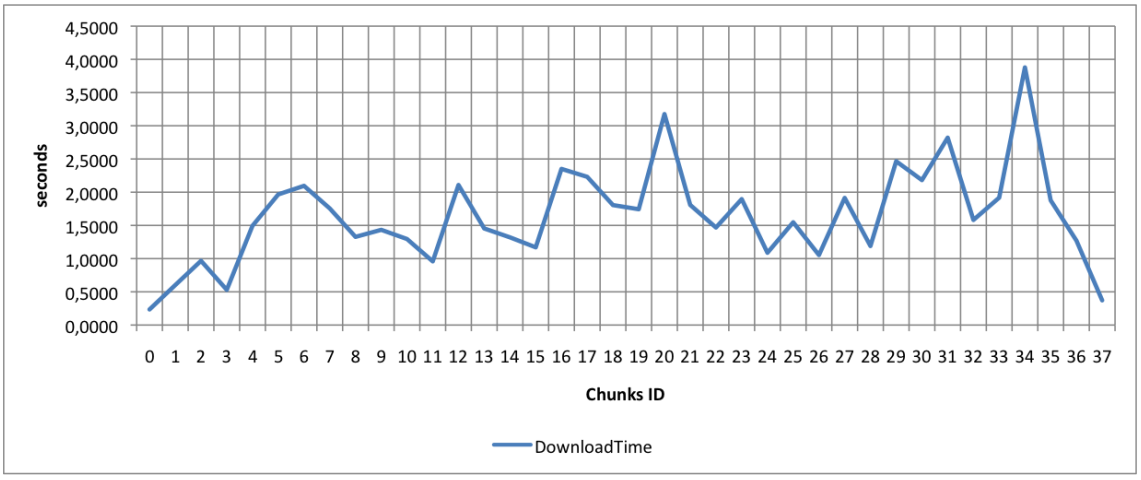

FIGURE 4.7 – Download Time during a demo playback

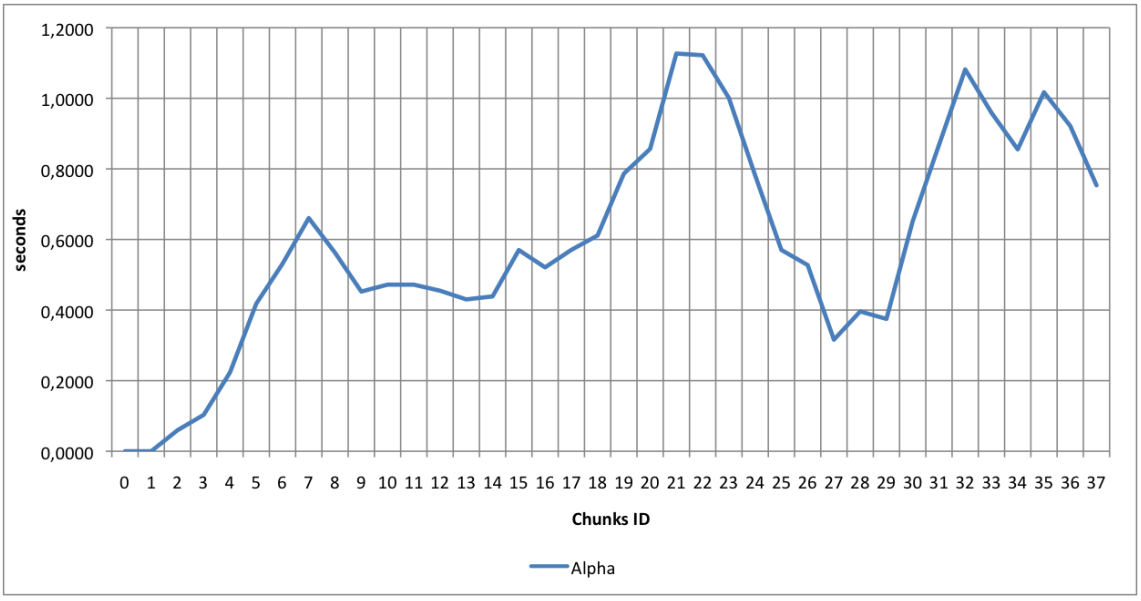

FIGURE 4.8 – Alpha variation during a demo playback

4.4 Conclusions of the experiments

The experiments showed that the solution described in this work works. Unfortunately, the used compiled version of FFMpeg is not optimized for a low CPU power usage and does not use hardware acceleration. It results in a too long frame decoding compared to the frame playback length.

It also showed that the partial implementation of a DRM Agent could be used for demonstration, but further tests should be done in order to evaluate the security level of such solution. Unfortunately, regarding the bad reputation of DRM systems, it seems unlikely that open-source or free post-installable DRM Agents could be available on the market.

Regarding the implementation itself, it seems that the heuristic is properly calibrated. On Figure 4.6, we can see that the bitrate increases until maximum quality. Figure 4.8 shows the Alpha variation. It progressively increases until reaching more than "1" on chunk 21. With this event, it starts lowering the video quality, which is visible on the scheme. After the crash, it starts again to increase quality which unfortunately causes another "too high" alpha, resulting in quality decrease.

Chapter 5

Conclusion

The purpose of this work was to design and implement a proof of concept for a new way of delivering protected content on mobile devices. The chosen target platform was Android which evolved a lot during all of this work (with five major releases). The technique to consume the media content was adaptive streaming and the protection mechanism was the OMA DRM 2.1.

All of those aspects were deeply studied through reading and understanding their documentation, meeting people working with them, building some proof of concepts, etc. In the end, a prototype was produced using lots of different technologies and tools widely available on the market such as FFMpeg, libCurl and OpenGL E.S. Yet, those tools are not optimized (except OpenGL E.S.) to run on a low power device as a mobile phone. But even then, the prototype was working and proved that it can successfully be implemented to create a commercial product with very few more efforts.

This work also emphasized a couple of weakness and strength of the used tools. First, Android is a very nice platform to develop for thanks to the power of Java. But unfortunately, when it's about to make some native development in order to use existing component, the provided tool, the NDK, is very poor which leads to a lot of problem when trying to compile and use existing libraries implemented for a desktop computer. In fact, compiling third-party libraries was the main issue when implementing the proof of concept, taking a lot of effort and time.

Another weakness revealed during this work was the general problem of a very poor documentation. For example, the FFMpeg documentation is, when existing, often incomplete or not clear at all.

The results of the prototype showed that the heuristic was working fine and switch quality when it is necessary, depending on the available bandwidth. It does not react too quickly when an accident occurs, but is still able to react on time. Unfortunately, as used libraries, here FFMpeg, are not optimized for mobile usage, it does not work (yet) quick enough to have a really "smooth" playback.

5.1 Further work

Even if the prototype is working, lots of work remains to do.

First of all, as the h.264 decoding is clearly too slow even with a pool of decoders ready all the time (to remove the latency caused by an initialization). Three approaches can be studied. The first one consist in re-compiling FFMpeg with only the necessary decoder, and optimize every possible parameter. The other approach would be to use more efficient codecs, such as CoreAVC [1], which seems to be more adapted to the mobile context. The last one would be to look into how to use the existing OpenCore framework on Android. Unfortunately, this framework is not available to developers but it is possible to natively use the installed OpenCore library (*libopencore.so*). This approach has the major drawback of being less portable as different versions of OpenCore may be installed on different devices.

Another interesting work would be to implement a fully OMA DRM 2.1 Agent for Android. This is a huge work due to the fact the platform does not support any mechanisms to do so (no secure file-system, very few cryptographic tools, etc.). Nowadays, there is no fully post-installable OMA DRM 2.1 Agent available for free or demonstration.

The existing code of the proof of concept probably require some optimizations regarding memory management, threads management and processes. It's probably possible to optimize the existing code in such a way that it would run faster.

The last big topic for further work would be to build a server software to do adaptive streaming for DRM protected content. To do that, two approaches are available. First, the server cuts the video chunks and protect it with DRM on the fly. It would probably lead to a serious overhead. The second approach would be to have a server which handles "pre-packaged" chunks. Those chunks would have been sliced and DRM protected once and for all. This approach seems to be the most efficient one.

1. http://corecodec.com/

Bibliography

[1] Ableson, F., Collins, C., and Sen, R. *Unlocking Android : A Developer's Guide*. Manning, 2009.

[2] Burnett, S., and Paine, S. *RSA Security's Official Guide to Cryptography*. RSA Press, 2001.

[3] CMLA Founders. *CMLA Client Adopter Agreement*. CMLA, 12 2009.

[4] Content Management License Administrator. *About CMLA*. http://www.cm-la.com/Default.aspx.

[5] Google Inc. *Android Open Source Project*. http://source.android.com/.

[6] Google Inc. *What is Android ?* http://developer.android.com/guide/basics/what-is-android.html.

[7] Haseman, C. *Android Essentials*. firstPress, 2008.

[8] Internet Information Service. *Smooth Streaming Technical Overview*. http://learn.iis.net/page.aspx/626/smooth-streaming-technical-overview.

[9] James, F. K., and Keith, W. R. *Computer Networking, a top-down approach*. Pearson Addison Wesley, 2008.

[10] Open Mobile Alliance. *OMA DRM Architecture Version 2.0.*

[11] Open Mobile Alliance. *OMA DRM Architecture Version 2.1.*

[12] Open Mobile Alliance. *OMA DRM Content Format Version 2.0.*

[13] Open Mobile Alliance. *OMA DRM Content Format Version 2.1.*

[14] Open Mobile Alliance. *OMA DRM Requirements Version 2.0.*

[15] Open Mobile Alliance. *OMA DRM Requirements Version 2.1.*

[16] Open Mobile Alliance. *OMA DRM Rights Expression Language Version 2.0.*

[17] Open Mobile Alliance. *OMA DRM Rights Expression Language Version 2.1.*

[18] Open Mobile Alliance. *OMA DRM Specification Version 1.0.*

[19] Open Mobile Alliance. *OMA DRM Specification Version 2.0.*

[20] Open Mobile Alliance. *OMA DRM Specification Version 2.1.*

[21] Open Mobile Alliance. *Open Mobile Alliance Digital Right Management Version 1.0.* http://www.openmobilealliance.org/technical/release_program/drm_v1_0.aspx.

[22] Open Mobile Alliance. *Open Mobile Alliance Digital Right Management Version 2.0.* http://www.openmobilealliance.org/technical/release_program/drm_v2_0.aspx.

[23] Open Mobile Alliance. *Open Mobile Alliance Digital Right Management Version 2.1.* http://www.openmobilealliance.org/technical/release_program/drm_v2_1.aspx.

[24] RSA Laboratories. *PKCS #7 : Cryptographic Message Syntax Standard*. http://www.rsa.com/rsalabs/node.asp?id=2129.

[25] RSA Laboratories. *Public Key Cryptography Standards #12 : Personal Information Exchange Syntax Standard*. http://www.rsa.com/rsalabs/node.asp?id=2138.

[26] Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., and Dolev, S. Google android : A state-of-the-art review of security mechanisms.

[27] Vila, E., and Borovka, P. Data protection utilizing trusted platform module. *ACM International Conference Proceeding Series Vol. 374* (2008).

[28] Zambelli, A. *IIS Smooth Streaming Technical Overview*. Microsoft Corporation, 03 2009.

[29] Zeng, W., Yu, H., and Lin, C.-Y. *Multimedia Security Technologies for Digital Rights Management*. Academic Press, 2006.

Annexe A

CMLA : Confidentiality and Integrity Table

Value	Confidentiality Required ?	Integrity Required ?	Consideration (Informative)
Device Private Key (DRM Agent Private Key)	Yes	Yes	Device Private Key (DRM Agent Private Key) is issued by CMLA and is implemented securely into a Device at its manufacturing time. Device must keep its confidentiality and integrity at all times.
Device Certificate (Chain) (DRM Agent Certificate (Chain))	No	No	Device certificate (DRM Agent Certificate) is issued by CMLA and is implemented into a Device at its manufacturing time.
Device Details	No	Yes	Device Details are manufacturer, model, and version information implemented in a Device at its manufacturing time. They are sent to RI in Device Registration Request within 4-pass Registration protocol. Device must keep its integrity at all times.
Trusted RI Authorities Certificate	No	Yes	Trusted RI Authorities Certificate (a.k.a. CMLA Root CA Certificate as defined in the CMLA Technical Specification) is issued by CMLA and is implemented into a Device at its manufacturing time. Device must keep its integrity at least until its expiry time.
Domain Context	-	-	RI sends Domain Context to a Device during 2-pass Join Domain Protocol. Device should keep this information at least until it leaves the domain. Device must keep confidentiality and integrity of the component information for the Domain.

Domain ID	No	Yes	Domain ID is sent to a Device by ROAP- JoinDomainResponse message. Device must keep integrity of the association between Domain ID and Domain Context information.
Domain Key	Yes	Yes	Domain Key is sent to a Device in Join Domain Response. Device must keep its confidentiality and integrity at all times.
Expiry Time	No	Yes	Expiry Time is sent to a Device in Join Domain Response.
RI public Key	No	Yes	Domain Context shall contain the RI public key for the case when the Domain Context Expiry Time extends beyond the RI Context Expiry Time. (DRM spec, 5.4.2.2.1)
RI Context	-	-	Device establishes RI Context with an RI through 4-pass Registration protocol. Device should keep this information at least until its expiry time. Device must keep confidentiality and integrity of the component information for the RI.
riURL	No	Yes	riURL is sent to a Device via ROAP- RegistrationResponse message.
Agreed protocol parameters	No	Yes	Agreed protocol parameters are shared between a Device and an RI by ROAP-DeviceHello and ROAP-RIHello sequence.
Protocol version	No	Yes	Protocol version is shared between a Device and an RI by ROAP-DeviceHello and ROAP-RIHello sequence.
Trusted Device Authorities	No	Yes	Trusted Device Authorities are sent to a Device by ROAP-RIHello message.
RI ID	No	Yes	RI ID is sent to a Device by ROAP-RIHello message.
Information whether an RI has stored Device Certificate	No	Yes	
OCSP Responder Certificate Chain (Public Key)	No	Yes	OCSP Responder Certificate is sent to a Device by RI's responses during 4-pass, 2-pass, 1-pass ROAP protocol.
Current (valid) OCSP response	No	Yes	OCSP response is sent to a Device by RI's responses during 4-pass, 2-pass, and 1-pass ROAP protocol.
RI Certificate Chain (Public Key)	No	Yes	RI Certificate Chain is sent to a Device by RI's responses during 4-pass, 2-pass, and 1-pass ROAP protocol.
RI certificate validation data	No	Yes	
Domain Name Whitelist	No	Yes	Domain Name Whitelist is sent to a Device by ROAP-RegistrationResponse message.
Expiry Time	No	Yes	
Replay Protection Cache	-	-	Device must have two kinds of replay protection caches and keep their integrity at all times.

with ¡GUID, RITS¿ entries	No	Yes	
with only ¡GUID¿ entries	No	Yes	
Device RO / Domain - RO	-	-	RI sends Device RO /Domain RO to a Device by ROAP-ROResponse. Domain RO may also be received by other methods.
Permission / Constraint	No	Yes	
Content Encryption Key	Yes	Yes	
Z	Yes	Yes	
Key Encryption Key	Yes	Yes	
Rights Encryption Key	Yes	Yes	
MAC Key	Yes	Yes	
Status information for Stateful Rights	No	Yes	Device must keep status information for each stateful RO and keep updating it when the associated content is consumed. Device must keep its integrity as long as RO is usable.
Transaction ID	No	No	
GroupKey	Yes	Yes	The GroupKey is included in the extended headers of a DCF within an OMADRMGroupID box.

Annexe B

Rights Object Sample

The rights depicted in this example grant unconstrained permission to play the corresponding DRM Content.

```
<o-ex:rights
        xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
        xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
        xmlns:oma-dd="http://www.openmobilealliance.com/oma-dd"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
        o-ex:id="C.1">

         <o-ex:context>
                <o-dd:version>2.1</o-dd:version>
                <o-dd:uid>RightsObjectID</o-dd:uid>
         </o-ex:context>
         <o-ex:agreement>
                <o-ex:asset>
                        <o-ex:context>
                                <o-dd:uid>ContentID</o-dd:uid>
                        </o-ex:context>
                        <o-ex:digest>
                                <ds:DigestMethod Algorithm="http://www.w3.org
                                    /2000/09/xmldsig#sha1"/>
                                <ds:DigestValue>DCFHash</ds:DigestValue>
                        </o-ex:digest>
                        <ds:KeyInfo>
                                <xenc:EncryptedKey>
                                        <xenc:EncryptionMethod Algorithm="http://
                                            www.w3.org/2001/04/xmlenc#kw-aes128"/>
                                        <ds:KeyInfo>
                                                <ds:RetrievalMethod URI="
                                                    REKReference"/>
                                        </ds:KeyInfo>
                                        <xenc:CipherData>
                                                <xenc:CipherValue>EncryptedCEK</
                                                    xenc:CipherValue>
                                        </xenc:CipherData>
                                </xenc:EncryptedKey>
                        </ds:KeyInfo>
                </o-ex:asset>
                <o-ex:permission>
                        <o-dd:play/>
                </o-ex:permission>
        </o-ex:agreement>
</o-ex:rights>
```

Listing B.1 – Rights Object Sample

Annexe C

Adaptive Streaming Client Manifest Sample

The following ISMC file is the one used with the demo content.

```
<?xml version="1.0" encoding="utf-8"?>
<!--Created with mod_smooth_streaming(version=1.0.8)-->
<SmoothStreamingMedia MajorVersion="1" MinorVersion="0" Duration="1191200000">
        <StreamIndex Type="video" Subtype="H264" Chunks="38" Url="QualityLevels({
            bitrate})/Fragments(video={start_time})">
                <QualityLevel Bitrate="363000" FourCC="H264" Width="336" Height="
                    192" CodecPrivateData="000000016764000
                    DAC2CC5054668400000030040000000CA3C50A65800000000168EEB2C8B0" />
                <QualityLevel Bitrate="228000" FourCC="H264" Width="256" Height="
                    144" CodecPrivateData="000000016764000
                    DAC2CC50404E84000000300400000CA3C50A65800000000168EEB2C8B0" />
                <QualityLevel Bitrate="135000" FourCC="H264" Width="192" Height="
                    112" CodecPrivateData="000000016764000
                    CAC2CC50C3E8400000300040000300CA3C50A6580000000168EEB2C8B0" />
                <QualityLevel Bitrate="103000" FourCC="H264" Width="176" Height="96
                    " CodecPrivateData="000000016764000
                    CAC2CC50B36840000030004000003 00CA3C50A6580000000168EEB2C8B0" />
                <QualityLevel Bitrate="85000" FourCC="H264" Width="112" Height="64"
                     CodecPrivateData="000000016764000
                    BAC2CC51C9A1000000300100000030328F14299600000000168EEB2C8B0" />
                <c n="0" d="40000000" />
                <c n="1" d="41600000" />
                <c n="2" d="28400000" />
                <c n="3" d="37200000" />
(...)
                <c n="36" d="40400000" />
                <c n="37" d="28000000" />
        </StreamIndex>
        <StreamIndex Type="audio" Subtype="mp4a" Chunks="38" Url="QualityLevels({
            bitrate})/Fragments(audio={start_time})">
                <QualityLevel Bitrate="95000" FourCC="mp4a" WaveFormatEx="
                    FF00020044AC0000E02E00000100100002001210" />
                <c n="0" d="40170521" />
                <c n="1" d="41563719" />
                <c n="2" d="28328345" />
                <c n="3" d="37151927" />
(...)
                <c n="36" d="40402721" />
                <c n="37" d="27631746" />
        </StreamIndex>
</SmoothStreamingMedia>
```

Listing C.1 – Adaptive Streaming Client Manifest Sample